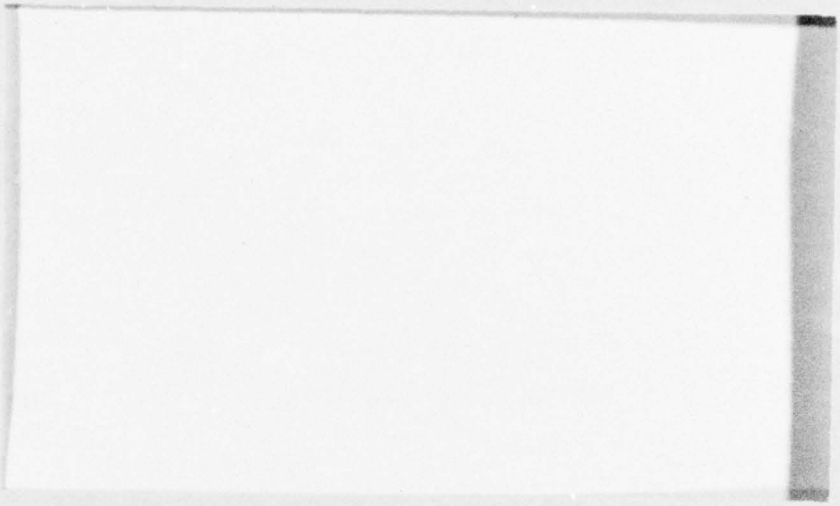
AD-A042 161 MARYLAND UNIV COLLEGE PARK DEPT OF COMPUTER SCIENCE F/G 9/4
OPTIMAL AND HEURISTIC SYNTHESIS OF HIERARCHICAL CLASSIFIERS.(U)
AUG 76 A V KULKARNI AF-AFOSR-2901-76
UNCLASSIFIED TR-469 AFOSR-TR-77-0825 NL

1 OF 2
AD A042161

COMPUTER SCIENCE TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742



(15) AF-AFOSR-2901-76,
NSF-ENG-73-04099 & AFOSR 76-2901

(12) 187 p.

(14) Technical Report TR-469

(11) August 1976

(6) OPTIMAL AND HEURISTIC SYNTHESIS OF HIERARCHICAL CLASSIFIERS.

(10) By
Ashok Vasa[illegible]rni

(9) Doctoral thesis,

(18) AFOSR

(19) TR-77-0825

(16) 2304

(17) A2

Dissertation submitted to the Faculty of the Graduate School of the University of Maryland in partial fulfillment of the requirements for the degree of Doctor of Philosophy, 1976.

This research was supported in part by the Mathematics and Information Sciences Directorate, Air Force Office of Scientific Research, Air Forcy Systems Command, USAF, under grant AFOSR 76-2901, and in part by the National Science Foundation, Control and Automation Branch, Engineering Division, under grant ENG 73-04099, to the Laboratory for Pattern Analysis, Department of Computer Science, University of Maryland, College Park, Maryland.

(14)

409022

ABSTRACT

Title of Thesis: Optimal and Heuristic Synthesis of Hierarchical Classifiers

Ashok V. Kulkarni, Doctor of Philosophy, 1976

Thesis directed by: Dr. Laveen Kanal, Professor of Computer Science
Department of Computer Science

Multistage schemes such as hierarchical classifiers have been found useful for many multiclass pattern recognition tasks. This dissertation investigates the theoretical properties of a general model of multistage multiclass recognition schemes. The generality of the model allows one to describe a large class of parametric and non-parametric schemes commonly used in terms of the model parameters. Two classes of admissible and optimal strategies for obtaining the optimal decision are analyzed. These strategies employ lower and upper bounds on a risk function to improve the search efficiency. New methods of computing the bounds are investigated for the cases when the features are class-conditionally statistically independent and where they satisfy a first-order tree dependence relation. Bounds are also derived for use in nearest-neighbor classification schemes employing a Euclidean distance measure and various similarity measures for non-metric feature vectors.

Hierarchical classifiers are special types of multistage recognition schemes wherein at each stage certain classes are rejected from consideration as labels of the test sample. Theoretical properties of decision trees whose node decisions are statistically independent are investigated. Even under this independence assumption the optimal tree design task is a complex one.

A three phase decomposition of the tree design problem is proposed viz. tree skeleton design, feature selection at its nodes and decision function design at each node. Optimal solutions to each design phase are

obtained using a dynamic programming formulation.

These optimal design methods rapidly become cumbersome in computational resources as the number of features and classes increase. This study proposes various techniques for reducing the computational complexity incurred in finding the optimal features to be measured at each node and the optimal decision policy. A method of clustering decision rules and rejecting sets of suboptimal rules without evaluating each individual one is proposed. Feature ranking and a branch-and-bound method are described for reducing the possible feature assignments to be considered in finding the optimal feature measurement policy.

In practice, the decision rules at the nodes have to be estimated from a finite set of design samples. This work investigates the relationship between the expected tree performance, sample size and the number of states (quantization levels) of each feature. It is shown that for an M-class recognition scheme using a decision tree, there exists an optimal quantization complexity. The optimum complexity increases with sample size and with the number of classes to be distinguished. For small sample sizes, it is shown that a multistage decision scheme can have a lower error rate than a single stage scheme which uses all the available measurements in an M-way decision rule.

ACKNOWLEDGEMENT

I wish to express my gratitude to Professor Laveen Kanal who inspired this research and guided and encouraged me during its development. My thanks also go to Dr. Ashok Agrawala for his various suggestions for improving the presentation of the material, and the many useful discussions we have had during these past few years. Finally, my heartfelt thanks go to my wife Ranjana, who encouraged me along the way.

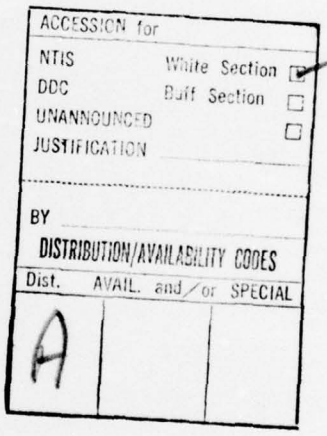


Table of Contents

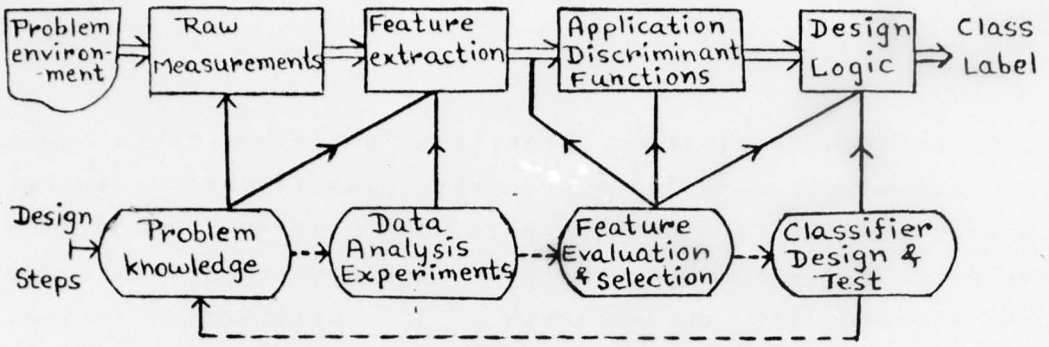
LIST OF TABLES

1. Introduction

In pattern recognition practise, decision trees have been extensively used for multiclass classification tasks. However, theoretical developments in pattern recognition have essentially avoided addressing the problem of designing such hierarchical classifiers. In practice, various heuristic and ad hoc methods are employed. This dissertation investigates the theoretical properties of a general model of multistage multiclass recognition schemes. The generality of the model allows one to describe a large class of parametric and non-parametric schemes proposed in the literature in terms of the model parameters. Hierarchical classifiers comprise an important special case of this model. We derive new theoretical properties of such classifiers and investigate the use of optimization methods for their design.

This chapter presents the background for the problem and the motivation which led to the investigation reported in subsequent chapters, and outlines the scope of this dissertation. The first section describes the components of a statistical pattern recognition system and the various design phases that guide its development. Section 2 outlines the trade-offs between three factors, viz., classifier performance, complexity, and measurement cost, that occur in the design of any practical system. The various multistage schemes proposed in the literature for achieving this trade-off are reviewed in section 3. The last section summarizes the contributions of this dissertation and its relationship to the existing literature on multistage multiclass recognition.

1.1. A System Configuration



⇒ Flow of data in operational system

→ Design influences

--→ Flow of information during the design cycle

Fig. 1.1

Most statistical pattern recognition systems can be divided into four phases (Fig. 1.1) through which an observed sample passes before it is classified[8]. These are (i) a measurement phase, (ii) a feature extraction stage, (iii) a phase during which a set of discriminant functions is applied to the feature vector extracted, and, (iv) a decision logic step that uses the outcomes of the functions to classify the sample.

The raw measurements consist of a potentially large set of measurements that can be taken from a sample and is guided by problem knowledge which suggests measurements of interest and the state of the art of transducer technology which dictates what measurements are feasible. Given this measurement set, certain features may be synthesized on the basis of prior information, (e.g., features such as the area and perimeter of a nucleus of a white blood cell), while

other features may be extracted from data analysis experiments conducted in an interactive mode. Linear and nonlinear mappings to one-space or two-space, and discriminant vector projections, fall into this latter category. During this data analysis phase, certain important information, such as the modes of a distribution (obtained via clustering experiments, for example) may be gathered and pas'sed on to the subsequent feature selection phase.

Having extracted a set of N features, the feature selection task consists of finding the 'best' subset of n out of N features which will optimize the classifier performance. Since the goodness of a set of features depends on the form of the classifier in which they will be used, the feature selection and classifier design tasks are strongly coupled.

The classifier design task consists of estimating parameters of the discriminant functions and designing the decision logic to optimize the classifier performance in the field. The entire design cycle may have available to it a set of labelled or unlabelled samples from which to glean the information it needs in each phase.

The implementation of such a system involves several iterations through the design cycle since design choices made at one stage affect all subsequent stages. While the feature extraction phase is problem dependent, the feature selection and classifier design steps can be automated.

1.2. The Design Tradeoffs

The design procedure of a practical pattern recognition scheme must contend, in general, with three counteracting factors, viz., the performance of the classifier, the complexity of the design, and the cost of taking feature

measurements. The components of these factors contributing to the design problem are depicted as a tree diagram, in Fig. 1.2 below.

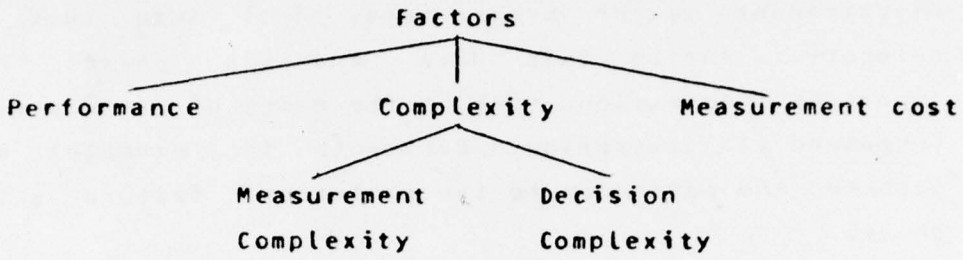


Fig. 1.2

The performance index of a classifier is usually taken to be the average misclassification rate, or average loss. If X represents a random sample (set of feature measurements), having a class conditional probability function, p(X/wi), d(X) is a decision rule that classifies X into one of M classes, w1,....wM, and C(wi,wj) denotes the cost (loss) incurred in classifying an wi sample into wj, then the average loss is given by,

$$R = \sum_{i=1}^{M} \int_{X} P(wi) \cdot C(wi, d(X)) \cdot p(X/wi) \quad \ldots\ldots\ldots (1.1)$$

where P(wi) is the a priori class probability of class wi.

In equation (1.1), X refers to a vector of features to be measured. There is a cost associated with taking new measurements, such as the cost of added instrumentation or computational resources. Sometimes this cost may be an intangible quantity. In medical diagnosis, for example, taking certain measurements may affect the patient's health,

or it may take several days to be performed, during which time the patient might remain hospitalized. This cost, referred to as measurement cost, is incurred whenever the decision rule, d(), requires the particular feature measurement. It is assumed in this discussion that classifier performance and costs, whether real or intangible, can be expressed in some common quantitative terms.

The phrase 'complexity of the classifier', is used here as a general term to describe two factors, (1) the measurement complexity, which depends on the number of features used and the possible states (values) that these can assume, or parameters needed to describe their distributions, and (2) the decision complexity, which depends on the form of the decision rule, d(X), i.e. the parameters needed to describe it. For a parametric classification scheme, the measurement complexity controls the number of parameters to be estimated, e.g. if N statistically independent features are used, and each feature is discrete, taking on one of m values, the total probabilities to be estimated per class are, $N.m^N$. For a given desired error rate, and a fixed feature set, the minimum decision complexity is a function of the 'ease' of separability of the classes in that feature space. In other words, if a sample vector, X, is regarded as a point in feature space, and if d(X) is a surface that partitions this space into disjoint regions, each having a class label,(i.e. the decision made if a sample lies in that region), then for a given error rate, e, the minimum number of parameters needed to specify the surface, d(X), would be larger if the classes were highly overlapped, and small if they were perfectly separable, or overlapping only slightly.

It has been shown [17,27] that the error rate can be

made arbitrarily close to zero if one had an infinite supply of features, where each feature contributed to the class discrimination. In practice, however, a zero error rate is often unachievable. First, the set of potential discriminatory features, though large, cannot be infinite, since in most applications, these are band-limited observations [17].

Secondly, in most cases one has to estimate the parameters of the assumed underlying distribution using a finite set of samples. It has been shown [17], that there exists a relationship between the optimum measurement complexity and the sample size, in that if any more or fewer than this optimum number of features are used in a classifier, its performance would be worse than the optimal value.

For a given set of features, the Bayes rule minimizes the average risk. However, for most problems of interest, the Bayes decision surface is too complex and would require excessive storage and computational cost. Hence, to reduce the classifier's design and running cost, this surface is often approximated by a simpler surface, e.g. a set of piecewise linear hyperplanes might be used to approximate a polynomial surface. The error rate using this approximation is larger than the Bayes error rate.

Finally, the cost of taking a particular measurement at a certain stage of the decision process might not be justified in view of the small improvement in performance it would give. Hence, this cost also restricts the performance of a classifier for certain applications.

Hence, measurement cost, decision complexity and the measurement complexity- sample size relationship, place an upper bound on the performance of a practical classification scheme. The interaction between these factors gives rise to

the feature selection problem, since one is forced to limit the number of features to be used in the classifier. The trade-off between performance and complexity gives rise to the classifier and decision rule design problems. These problems are accentuated for multiclass and multimodal classification tasks. We shall briefly describe the various feature selection methods proposed in the literature, and point out some of their limitations when applied to a multiclass discrimination problem.

1.2.1. Feature Selection

Feature selection, as usually discussed in the literature, is the problem of finding the best set of n out of a total of N available features, that will optimize the performance of a classifier. This performance is assumed to be the Bayes error rate. Where this is difficult to compute, various loose bounds on this error are used . The best set of n features can always be found by examining all $\binom{N}{n}$ combinations. However, for large N one may want a more efficient search strategy that finds a good though not necessarily optimal subset of features. The search for the optimal set is complicated by the fact that the best k out of N features need not be contained in the best k+1 out N features, even if the features are statistically independent [35]. The different feature selection methods differ in the criterion function sought to be optimized, and the method of search.

Let G denote the function that evaluates the feature set goodness. Two types of G functions have been proposed. The first type are various distance and information measures which are loosely related to the Bayes error. A list of such measures appears in [8]. All these functions involve the class-density functions which may have to be estimated . The

second category of G functions may be defined as geometric measures. Some of these are described in [21]. Functions of between-class and within-class scatters, fall in this category.

The search methods may be divided into four groups, (i) single step selection, (ii) without-replacement method, (iii) iterative refinement method, and (iv) the recursive method.

In the single-step method, the n features are chosen in one step, e.g. the n features which contribute the largest amounts to the eigenvectors corresponding to the m largest eigenvalues of the covariance matrix [21]. Other methods are contained in [29,30].

The without-replacement policy consists of adding to the subset of k features already chosen, that feature which maximizes the criterion function, G. Here G could be a goodness measure of the single feature, or a measure of the correlation of that feature with the k features, or a weighted sum of the two, etc. [20]. In multiclass schemes, certain modifications of these methods have been suggested, [20],e.g. the k+1 th. feature may be chosen as that which best separates the class pair most confused by the k feature set.

The iterative refinement policy consists of replacing some m out of the k features chosen at the k th. step, by some other features, until no improvement in G is possible. Then the k+1 th. feature which causes the largest increase in G is added to this refined set, and the search proceeds to the k+1 th. stage.

The recursive method, also called a dynamic programming method [3], may be described in the following manner. At the k th. step we have found the 'best' k out of N features. To find the best k+1 features, evaluate all k+1 feature sets

obtained in the following two ways; (i) the best k feature set found along with any feature from the remaining feature set (without replacement policy), and, (ii) any set of m out of the k feature set together with the best set of (k+1-m) features out of the remaining N-k features, the latter being defined recursively in a similar manner. The best k+1 features out of N, are the features evaluted in (i) and (ii) with the largest G value.

The above methods of selecting features have some drawbacks when used in a multiclass recognition system. Most of the geometric measures of goodness are inadequate if more than two classes have to be distinguished, or if the classes are multimodal. For multiclass and multimodal problems, a single geometric measure usually cannot describe the goodness of a feature set. By using a single criterion function, G, it is being assumed that a Bayes classifier will be used to discriminate the classes. However, from the earlier discussion, it is seen that a Bayes classifier may be too difficult and complex to implement. Thus, these feature selection methods do not connect the selection task with that of specifying the form of the classifier in which they would be used. Another factor to be considered is that of optimum dimensionality. For a given sample size per class, one may not want to use more than say n features. However, the best n features to discriminate one class pair, may be different from the best set to distinguish between another pair. By assuming that some n features are used in a single step to make an M-way decision (for an M-class problem), one is forced to compromise the feature set choice. A better use of the features might be to split the decision into several stages, where at any stage one set of classes is distinguished from another. Thus, at each step, one could use the n features best suited for that task.

Another limitation of the above search methods is that they do not take into account the measurement cost-performance tradeoff that has to be made in many applications.

Multistage schemes have been proposed in the literature to achieve the performance-complexity-cost tradeoffs that must be made in practice. In such schemes, a sequence of feature measurements are taken from the sample, and a sequence of 'guesses' made regarding the sample label. The various schemes proposed differ in the strategy used to select the next feature to be observed, and the decision function used to refine the 'guess' of the label. By breaking the multiclass decision making problem into a series of less complex decisions, the complexity in the design of each such decision is reduced. The feature to be measured at any stage, can be chosen depending on the classes to be distinguished. Moreover, if the cost of taking the next measurement is unjustifiable in view of the gain in accuracy, the measurement process can be terminated, and a 'good' decision made on the basis of observations taken.

1.3. Literature On Multistage Classification

Multistage decision schemes and hierarchical classifiers have been extensively used to solve recognition problems and experience with such methods is discussed in several papers. This body of work can be grouped into three broad categories:

(1) Conversion of decision tables to optimal decision trees,
(2) Sequential pattern classification methods, and
(3) Hierarchical classification methods.

There has been a considerable amount of work reported in

converting decision tables into optimal decision trees [9,15,24,25,26]. In these cases, the criterion of optimality is the average number of nodes traversed to classify a sample, or the total number of nodes in the tree[25]. However these table conversion methods do not address the problem of trading efficiency(as measured by the average number of nodes traversed) for misclassification cost, since they assume that the patterns can be unambiguously distinguished. Knuth[9] considers a dynamic programming approach to construct an optimal binary search tree for alphabetical key words, where at each node, one stores a key word, and the test at that point consists of seeing if the sample key matches that key, is less than that key or greater than it. Meisel and Michalapolous[10] discuss the use of a recursive procedure to arrange a set of piecewise constant boundaries in metric space, so as to minimize the average number of comparisons needed to classify a sample. However, they assume that the feature space has already been broken up into the various decision regions. Thus, the algorithm rearranges the order of the tests optimally, without affecting the misclassification rate.

Stoeffel[13] and Bell[15] use a two phase method for designing a hierarchical classifier for character recognition applications. The basic features are binary (e.g. presence or absence of some structural component). In the first phase, these features are used to synthesize more complex features which are basically binary vectors with 'dont care' bits wherein each bit represents the presence or absence of a given property.This collection of vectors (called prototype vectors) are used to generate a decision table for the multicharacter recognition problem. In the second phase, the decision table is converted into a decision tree having a minimum path length. Bell[15] gives a

heuristic algorithm for deriving the prototypes (which he refers to as decision rules). The 'decision rules' for each class are generated automatically using design samples. The procedure starts with the most general rule with all don't care bits, i.e. accepts all samples. Then the bits are sequentially specified, and the merit of a rule is measured by,

$$T \cdot p - U \cdot q$$

where, p, q are constants determined empirically and varied during the procedure in a certain way, T is the number of samples from the target class which the rule accepts, and U is the number of non-target samples accepted by it.

One of the early investigations of sequential methods for two class problems was carried out by Wald [28]. This method assumes that an infinite series of measurements can be taken sequentially from the test sample. After taking each measurement, the ratio of aposteriori likelihoods of the two classes is compared with two threshold values. If the ratio falls between the two, the measurement process is continued, while if it falls to one side or the other of both thresholds, the sample is classified into one or the other class. A generalization of this method for the multiclass case was presented in [5]. It can be shown that for the two class case, the Wald sequential ratio test requires the minimum average number of measurements for a given error rate per class.

Fu [5] discusses the problem of optimal sequential decision making and feature ordering when the number of features is finite. He uses a dynamic programming formulation to compute the optimal policy for a fixed feature ordering, and the optimal feature ordering and

decision policy, for a given number of features, N. This method consists of starting at the last stage of the decision making process and evaluating the optimal policy for all possible histories of measurements. The decision at that point is that of either classifying the sample into one of M classes, or taking the next measurement. This choice is based on the minimum of the risk that would be incurred if the classification were made, and the cost incurred if the measurement process were continued. Thus, the decisions computed at each stage optimize the weighted sum of risk and measurement cost for any sample that reaches that stage.

Hart[18] considers the problem of finding an admissible strategy (Bayes optimal) for a probabilistic decision tree.The tree is binary, and each terminal node represents a joint hypothesis comprised of the subhypotheses denoted by the arcs on the path to that terminal. Thus each measurement provides an aposteriori likelihood of each of a pair of subhypotheses. Under the assumption that the likelihood of a composite hypothesis depends only on the measurements taken along that path, and the assumptions that the measurements are conditionally independent and independent, Hart shows that the aposteriori likelihood of the joint hypotheses below a node can be used as a heuristic to order the nodes which are candidates for traversal. The resulting algorithm is admissible and optimal.

Fukunaga[6] has proposed the use of branch-and-bound methods for reducing the distance computations in nearest-neighbour schemes. His method consists of making a hierarchical partitioning of the design sample set and storing at each node of the hierarchy the mean vector and radius of the samples represented by that node. These two parameters are used to put a lower bound on the minimum distance between a test sample and any sample in the design

set at the node. If the current minimum distance of the test sample from any design sample is less than this lower bound, the entire set of design samples at that node can be discarded from consideration as nearest neighbours. In this way, the nearest neighbour can be found with a smaller average number of distance computations.

The methods proposed in the literature for designing hierarchical classifiers which minimize the sum of measurement cost and risk, are mostly top down heuristic methods which are suboptimal solutions to the problem, [4,14,16].Mattson and Dammann[16] consider the use of linear discriminants to detect and code the clusters in a multiclass problem, wherein the node decisions are binary and the thresholds are set manually by inspecting the scatter of the samples(labelled) along that axis(the Fisher direction). Friedman[4] describes a nonparametric classification method that consists of splitting a set of labelled design samples into successively smaller sets, until all samples in each of the terminal sets belong to the same class, or have a membership less than some specified constant. The feature chosen at each node to split the samples is that which has the largest value of the Kolmogorov variational distance between the two class populations projected along that feature axis, or along the Fisher direction using all features.The threshold at that node is selected to maximize this variational distance. For multiclass schemes, two methods are suggested. The first uses M such trees where the i th. tree separates the i th. class from the other classes. The test sample is classified by passing it down all M trees and labelling it with the class which has the largest number of samples in the 'buckets' (terminal nodes) into which the test sample falls. The second scheme uses a generalization of the Kolmogorov

distance, and a single tree to make the classification. The generalized distance is the variance of the the M estimated cumulative distribution functions along the particular feature axis. The feature which has the largest variance value is the feature chosen to split the sample set.

Wu [14] describes a top down heuristic method for the design of a hierarchical classifier. Starting with the design samples from all the classes, the procedure successively partitions the samples (classes) using a non-supervised clustering algorithm, and evaluating all 'likely' feature subsets. The feature subset chosen for the node, di, is that having the smallest value of the evaluation function, which is of the form,

$$E(di) = -T(di) - K.e(di) + \sum_{j=1}^{c_i} E(dl+j)$$

In the above equation E is the evaluation function, e(di) is the error rate at di using that feature set, T(di) is a measure of the time needed to make the decision at di, K is a weighting term, and ci are the number of descendant nodes of di, obtained via the clustering procedure. E(dl+j) are the evaluation functions of the descendant nodes, and since these nodes have not yet been expanded, E(dl+j) is assumed to be a sum of two terms, viz.,

$$E(dl+j) = -T(m,nj) - K.C$$

where T(m,nj) is the computation time at that node, assumed to be a function of the total set of features, m, and the number of classes, nj, in that cluster. The error at dl+j is assumed to be a constant, C. Thus, this top down procedure of tree generation consists of splitting di using the feature set with the smallest value of the evaluation function F(di), and then repeating the process on each of

the descendant nodes, until each terminal set contains samples from a single class.

1.4. Scope Of This Research

The preceding discussion of the reported work in multistage classification highlights the following factors:

(1) Most of the work on decision trees has dealt with the task of converting decision tables to optimal trees. These studies assume that the pattern classes are perfectly separable, and hence unambiguously separated by the decision table. Therefore, they seek to minimize the number of tree nodes, the average path length, or some combination of these two factors. In statistical pattern classification, however, the classes do 'overlap', and the error rate of the tree is an important consideration. The crucial part of the design is that of designing the decision table, since this determines the error rate. Moreover, for finite design samples, one has the added complexity of selecting the features which will be used in the classifier. The table conversion algorithms do not address themselves to these two problems viz., feature selection and misclassification error.

(2) There has been some work done on designing optimal sequential recognition procedures for multiclass problems. Sequential methods of recognition differ from hierarchical classifiers both in the ordering imposed on the sequence of feature measurements and the ordering on the set of possible class labels. Sequential schemes impose a linear ordering on the features, and in most cases there is no particular order on the class labels, i.e. any class can be accepted at any stage of the measurement process. In hierarchical methods,

the features as well as the class labels are order hierarchically. Thus at each step of the measurement, some classes are rejected from consideration as candidates for the test sample's label. Dynamic programming has been used to find the optimal decision policy and feature ordering for sequential schemes. This work has not been extended further perhaps because of the exponential growth in computing required for designing such schemes, as the number of features and classes increase.

(3) The methods of designing hierarchical classifiers reported in the literature are heuristic methods making no claims to optimality even under restrictive assumptions. There has been little study done on the applicability of optimization methods such as dynamic programming and branch-and-bound techniques for the design of such classifiers.

This research was aimed at developing theoretical approaches to the analysis and synthesis of a broad class of multistage recognition schemes, including hierarchical classifiers, and studying the use of optimization methods for their design.

Chapter 2 develops a general theoretical model, a state-space model, to describe the behaviour of multistage schemes. It is shown that most of the methods proposed in the literature can be described in terms of this model. The model's generality allows one to define new types of classification schemes. In this model, a state consists of a measurement set, and a set of possible classifications of the sample. An edge in the graph represents the action of observing a particular feature(or a set of features), and has a cost, the measurement cost, associated with it.A goal

state is any state which contains one or zero class labels as the set of possible classifications. Depending on the manner of definition of decision making costs, two types of admissible strategies are defined, viz., an S-admissible strategy, and a B-admissible strategy. The goal cost is assumed to be a weighted sum of measurement cost and misclassification risk.Optimality and admissibility of the strategies can be proved under certain conditions.

The heuristics used in such searches employ bounds on the misclassification risk, to decrease the nodes to be searched in the state-space graph. Methods of computing bounds are derived for several parametric and nonparametric classification schemes. This general model has the advantage that nonparametric schemes, such as the nearest neighbour rule, using Euclidean distance, or similarity measures for nonmetric features, can also be modelled as a state space search. New bounds on such distance measures are derived, and it is shown that these bounds can lead to a substantial reduction in the number of distance computations needed to find the nearest neighbour.

In the above state-space model, a state consists of any subset of features, and any subset of class labels. If all states and all possible state transitions are considered, the model rapidly becomes too large and complicated to handle. One way to restrict the states, and state transitions, is to make the state-space graph explicit, i.e. to explicitly define the possible ordering of feature measurements. Such a graph can still be searched using an S-admissible or B-admissible strategy. A hierarchical classifier is a particular type of graph in which a single path is followed to classify a test sample, and where at each stage in the decision making process, some classes are rejected from consideration as possible labels of the

sample.

Chapter 3 derives a number of properties of hierarchical classifiers, and examines in some detail, those classifiers whose node decisions are statistically independent. Bounds are derived for the error in tree performance when such an assumption is made. It is also shown that when the independence assumption is valid, an upper bound on the total tree performance can be derived in terms of the performances of the classifiers used at the nodes of the decision tree. The tree design problem is complex even under this independence assumption. Two 'negative' properties bear out this fact. First, it is proved that optimizing the performance at each node does not necessarily optimize the total tree performance. Secondly, choosing at each node, that feature which makes the node decision with the least error, does not necessarily constitute the best choice of features to be used at the tree nodes.

The tree design problem can be simplified by using a three phase approach proposed in chapter 4. It investigates the use of dynamic programming in solving the following three design tasks:

(1) Design of the optimal policy at each tree node, given the tree structure, i.e. the tree skeleton and the features to be used at each node.
(2) Design of the optimal feature measurement and decision policy, given only the tree skeleton, i.e. the classes rejected at each node and the node hierarchy.
(3) Design of the optimal tree structure, i.e. the skeleton and the choice of features to be measured at each node given that at each node, the decision is a maximum likelihood rule using prior class

probabilities.

While the dynamic programming algorithms for solving (1) and (2) are optimal for any feature distribution, the algorithm presented to solve problem (3) is optimal only under certain conditions.

The dynamic programming methods rapidly become cumbersome in storage and computational requirements as the number of features and classes to be considered, increase. Chapter 5 studies methods of reducing the measurement and decision complexity at the expense of degraded performance. This chapter describes methods of reducing the complexity in determining the optimal decision policy at each node, and the optimal feature to be measured at each node.It proposes techniques for clustering decision rules when the features used at the tree nodes are discrete and non-metric. A branch-and bound algorithm for discarding rules which are known to be suboptimal is also suggested. The problem of assigning features to the nodes of a given tree skeleton is also considered. A dynamic programming formulation, and a branch-and-bound method for solving this problem are described. The use of feature ranking at tree nodes is an additional method of reducing the computations involved in tree design.

Chapter 6 explores the relationship between sample size, the number of classes, and dimensionality, as it affects tree design. In particular, it is shown that for small sample sizes, by breaking the M-class problem into a hierarchy of two-class problems, the degradation in the estimated performance from the true performance of the classifier, can be reduced. This result is illustrated for the case of discrete features. An expression for the mean accuracy of a decision tree for an M class problem is

derived. This analysis allows one to study the behaviour of the classifier performance as a function of sample size, quantization complexity and the number of classes, M. The existence of an optimum quantization complexity for a given sample size, discovered by earlier researchers[33,34] for a two-class problem, is shown to hold for an M-class decision tree recognition scheme.

In summary, a general model of multistage multiclass recognition schemes, which trade accuracy for cost has been formulated and analyzed. Systematic methods for the design of hierarchical classifiers have been investigated, as these represent important special cases of multistage schemes. This research has led to new results and new insights on multistage classification, and proved the usefulness of optimization methods in designing hierarchical classifiers.

2. General Model of Multistage Classification

This chapter presents a formulation of the multistage multiclass pattern recognition problem as a state space search. Two classes of strategies, called S-admissible and B-admissible strategies, are presented as being optimal for two types of pattern classification tasks.

An S-admissible strategy finds the minimum cost goal(node) in the state-space graph when the goal cost depends only on the features measured on the path to that node in the graph. The search strategy uses a heuristic function to decide the node to be traversed next, and terminates when a goal node is reached.

A different type of strategy is needed when the goal cost depends upon ALL features measured on the test sample. The Bayes risk is an example of such a cost function. The concept of a B-admissible strategy is defined for this kind of a search problem. A B-admissible algorithm's execution sequence can be regarded as composed of two parts, the first of which uses a heuristic function to decide the node to be traversed next, while the second part uses an upper bound on the goal cost to reject certain classificatory decisions. The algorithm terminates when all classes are rejected except one, this being the B-optimal goal. In some situations, it could terminate with the reject class, i.e. none of the classes is accepted. A theorem on the optimality of B-admissible algorithms is presented.

Viewing the multistage classification problem as a state space search allows one to describe a large class of parametric and non-parametric methods in a single framework. The search efficiency of the S-admissible and B-admissible strategies depends on the 'tightness' of the bounds on goal costs used by them to evaluate alternative search paths.

Techniques for computing upper and lower bounds on a goal cost, for use in admissible search strategies, are derived for the following special cases:

- Statistically independent discrete features.
- Tree dependent discrete features.
- Nearest neighbour classification using Euclidean distance (nonparametric).
- Classification using similarity measure between binary vectors (nonparametric).

2.1. State-Space Model

A multistage multiclass decision making process can be modelled as a search for a minimum cost goal node in a state-space graph, G, which is a 7-tuple,

$$G = (S, E, F, W, c, r, T)$$

where,

S : is the set of states(nodes) in the graph,

E : is the set of possible transitions(edges) between states in S,

F : is the total set of features,

W : is the total set of class labels, W=(1,2,...M), where M is finite,

c : is a non-negative real-valued cost function on the edges in E, and represents the measurement cost,

r : is a real-valued function on the 'goal' nodes in S and represents the misclassification loss incurred by making the classificatory decision associated with that node, and,

T : is the decision strategy used for deciding the

node to be traversed next; it could be a function of all measurements taken on the test sample.

A state (node) $s \in S$, is a tuple, (Fs, Ws), where,

$Fs \subset F$ is a subset of the features that are measured when that node is traversed, and,

$Ws \subset W$ is a subset of the class labels, and denotes the possible classifications that can be made on any path in the graph passing through s.

A goal node, s, is one for which,

$Fs = \emptyset$ and $|Ws| < 1$, i.e. only one classificatory decision is possible, or a 'reject' decision.

The observed values of the feature set Fs are random variables. The decision making process consists of starting with the initial node of G, taking the feature measurements associated with that node, and using the strategy, T to decide which of the successors of the node to traverse next. This process is repeated for any node selected for 'expansion'. If $N(s^*)$ denotes the set of states (nodes) on the path to a goal node, s^*, and $c(N(s^*))$ is the sum of arc costs along that path, and $r(s^*)$, the goal risk, then the total cost of making the decision s^* is defined by,

$$f(s^*) = c(N(s^*)) + r(s^*) \quad \ldots\ldots\ldots\ldots\ldots\ldots (2.1)$$

Based on the form of the risk, $r(s^*)$, one can define two broad categories of multistage classification schemes:

- the risk r is of the form, $r(s^*/Xs)$, where Xs is the set of measurements on the path to s^*.

- the risk r is a function of ALL measurements, not just those on the path to s*. If k denotes a stage variable and Xk the features observed till stage k, then r is of the form r(s*/Xk) and it could vary as more features are observed, possibly on other paths that don't lead to s*.

The following sections describe admissible strategies for each of the above types of classification schemes.

2.2. S-admissible Strategies

An S-admissible strategy is defined as one which, among all possible goal states in G, terminates with the goal state s*, for which the value of f(s*) is minimum, where,

$$f(s^*)=c(N(s^*))+r(s^*/Xs) \quad \text{.....(2.2)}$$

and Xs are the measurements on the path to s* in G.

A search strategy is defined in terms of an evaluation function which it uses to order the set of nodes, currently available for 'expansion' (i.e. the set of 'open' nodes). This evaluation function, f(n), for node n, is defined as,

$$f(n)=g(n)+h(n)+l(n)$$

where g(n) is the cost from the initial node to node, n, h(n) is an estimate of the arc cost from n to a goal node accessible from n, and l(n) is an estimate of the risk of a goal node accessible from n. Algorithm S described below, uses the function f to select the order of expansion of the nodes in G. This algorithm is similar to the ordered search method proposed in [12] but differs from it in that we also use a lower bound on the decision risk in the evaluation

function.

2.2.1. Algorithm S

Let OPEN refer to a set of OPEN nodes, i.e. candidates for expansion. Let CLOSED refer to the set of nodes already traversed.

Step 0: Set OPEN to contain the initial state, (λ, W).
Set CLOSED to the null set.

Step 1: For each node n in OPEN compute f(n), as,

$f(n) = g(n) + h(n) + l(n)$

where,

$g(n) = c(N(n))$

$h(n) = 0$ if n is a goal node,

$\leq \min_{j \in Wn} [\, c(N(j)) - c(N(n))]$ if n is not a goal.

$l(n) = r(n/Xn)$ if n is a goal node, Xn being the features observed on the path to n,

$\leq \min_{j \in Wn} [\, \min_{Y} r(j/Xn, Y)\,]$ when n is not a goal,

j is a goal below node, n, Y the set of features not yet observed and Wn the set of goal states accessible from n.

Step 2: Let n* be the node having minimum f(n).
Remove n* from OPEN and put it in CLOSED. If n* was a goal node, STOP with n* as the optimal goal else take the measurement x associated with n* and put the successor nodes of n* in OPEN, then repeat step 1.

Theorem 1: Algorithm S is S-admissible.

Proof: The optimality of a goal node n* put in CLOSED by algorithm S, follows from the fact that the evaluation function, f(n) underestimates the cost of ANY goal node below n. Since the value of f(n*) is less than f(n) for any OPEN node, it follows that n* has a smaller value of f() as given by equation (2.1) than any other goal node. Hence, it is optimal.

Theorem 2 shows that the lower bound on the risk of a goal is a non-decreasing function on the sequence of observations taken along the path to that goal. This fact is used along with a consistency assumption on the h function, to prove in Theorem 3 that the evaluation function, f(n) is non-decreasing on the sequence of nodes expanded by algorithm S.

Theorem 2: Let Xn be the measurements on the path to a goal node, n. Let X1, X2 be subsets of Xn, such that $X1 \subseteq X2$. Let $\bar{X}$ denote the complement of X with respect to Xn. Then,

$$\min_{\bar{X}_2} r(n/X2,\bar{X}2) \geq \min_{\bar{X}_1} r(n/X1,\bar{X}1)$$

Proof: Let $X3 = X2 \sim X1$, i.e. the set of features in X2 and not in X1. Then,

$$\min_{\bar{X}_2} r(n/X2,\bar{X}2) = \min_{\bar{X}_2} [\, r(n/X1,X3,\bar{X}2) \,]$$

$$\min_{\bar{X}_1} r(n/X1,\bar{X}1) = \min_{\bar{X}_2} [\, \min_{X_3} r(n/X1,X3,\bar{X}2) \,]$$

Since, $\min_{X_3} r(n/X1,X3,\bar{X}2) \leq r(n/X1,X3,\bar{X}2)$ for any particular value of X3 on the right hand side of the above inequality, it follows that,

$$\min_{\bar{X}_1} r(n/X1,\bar{X}1) \leq \min_{\bar{X}_2} r(n/X2,\bar{X}2)$$

Definition[12]: The heuristic function, h(n), which

estimates the arc cost from n to a goal node, is said to be consistent, if, for any two nodes, i,j, such that there is a path from i to j of cost, c(i,j),

$$h(i) - h(j) \leq c(i,j).$$

Theorem 3: If the function h used by Algorithm S is consistent, then the evaluation function, f(n) is non-decreasing on the sequence of nodes expanded by S.

Proof: Consider two nodes, i, j such that S closed i before j. There are two cases that could arise, either j is a descendant of i or it is not. If it is a descendant,

$$g(i) + h(i) \leq g(j) + h(j)$$

by the consistency assumption on h, since,

$$h(i) - h(j) \leq g(j) - g(i) = c(i,j) \quad .$$

Also by Theorem 2, $l(i) \leq l(j)$, hence, $f(i) \leq f(j)$ and the theorem is proved. If j was not a descendant of i, it means that when i was closed by S, there was an ancestor of j, say k, which was in the OPEN list and was not closed. Hence, $f(i) \leq f(k)$ since i was chosen. However, since k is an ancestor of j, by the earlier part of this proof,

$$f(k) \leq f(j).$$

Hence, it follows that,

$f(i) \leq f(j)$ and the theorem is proved.

The above theorem will be used to prove the optimality of algorithm S. The optimality property states that algorithm S will not expand any more nodes than any other S-admissible strategy which is 'less informed' than it.

Theorem 4: Consider two S admissible strategies, which use functions, h,land h', l', respectively, where h and h' are consistent, and such that,

$l(n) = l'(n)$ for all goal nodes, n, and,

$h(n) + l(n) > h'(n) + l'(n)$

for all other nodes in G.

Then the strategy using h', l', expands all the nodes expanded by the strategy using h and l.

Proof: Assume the contrary is true, i.e. assume there is a node i expanded by the strategy using $\{h,l\}$, which was not expanded by that using $\{h',l'\}$. Both algorithms terminated with the same goal node, say n*. By Theorem 3,

$g(n^*)+l(n^*) \geq g(i) + h(i) + l(i)$.

Since $l(n^*) = l'(n^*)$, n* being a goal node, and since,

$h(i) + l(i) > h'(i) + l'(i)$

it follows that,

$g(n^*) + l'(n^*) > g(i) + h'(i) + l'(i) = f(i)$.

However, by the assumption that strategy (h',l') didn't expand i, it implies that,

$f(i) < f(n^*)=g(n^*) + l'(n^*)$,

which is the converse of the previous inequality. Thus, there could be no node, i, expanded by $\{h,l\}$ and not by $\{h',l'\}$.

2.2.2. k-Step Lookahead Heuristic

In a particular classification scheme, the state-space graph, G, may be known explicitly. i.e. the possible orderings of feature measure- ments and decision making hierarchy are given. The graph is also finite since the feature set and the class label set are both finite. One can then define a 'k-step lookahead' heuristic, $D_k(n)$, for an open node n, as a lower bound on the additional cost incurred in going up to k levels beyond n in the graph.

Let,

Xn be the measurements taken on the path to n,

l(n/Xn) be a lower bound on the risk of any goal n* accessible from n, given that Xn has been observed,

c(i,j) be the cost of the arc joining nodes i and j,

T(i) be the set of successor nodes of i,, and

Gk(n) be the subgraph whose node set consists of n and all unopened nodes k or less arc lengths away from n, and whose edge set comprises all arcs in G joining nodes in this node set.

Then the heuristic Dk(n) may be computed using the following recursive procedure [Ref. 2,pp. 229].

$$Dk(i) = \min_{j \in T(i)} [c(i,j) + Dk(j)]$$

where, for a terminal node, j, in Gk(n),

Dk(j) =0 if j is not a goal node,

=l(j/Xn) if j is a goal node.

Lemma 1: If m > k, then Dm(n) $\geq$ Dk(n) for all nodes, n. The proof follows from the additive nature of arc costs.

Corrolary 1: If two S-admissible strategies use the evaluation functions,

f(n) =g(n) + Dm(n) and f(n) = G(n) +Dk(n), respectively, and m > k, then the strategy using Dk will expand all the nodes expanded by that using Dm.

Proof: The corrolary follows from Lemma 1 and Theorem 4.

2.3. B-admissible Strategies

An S-admissible strategy terminates when it first puts a goal node on the CLOSED list. It is guaranteed that this goal is optimal, because of the assumption that no measurements taken on other paths , not leading to this goal, can change the goal risk,r. However, if r were to change with additional measurements taken on other paths, then the optimality of the first goal node put into CLOSED cannot be guaranteed, because these observations may increase that goal's risk while decreasing the risk of some other goal. The question arises: if r is a function of all measurements(denoted by X), is there a search strategy which finds the goal node n* for which f(n*) is minimum, without taking all measurements, where,

$$f(n^*) = c(N(n^*)) + r(n^*/X) . \quad \ldots\ldots\ldots\ldots\ldots (2.3)$$

This section describes certain cases when such a strategy, called a B-admissible strategy, can be formulated. A B-admissible strategy uses an upper bounding function, u, defined in the following way.

Definition: Let Xk be the observations taken up to the k th. stage of a search, and let n be a goal node in G.Then an upper bounding function, u, is defined as,

$$u(n/Xk) \geq \text{Max } r(n/Xk, \bar{X}k)$$

where $\bar{X}k$ is the complement of the observations Xk, with respect to the total set of available observations, X.

Algorithm B described below, uses an upper bounding function u, and a lower bounding function analogous to that

used for S-admissible strategies, to obtain the B-optimal solution, n*. This algorithm is similar to Algorithm S except that it does not terminate when a goal node is put in the CLOSED list. Rather, at each iteration after a goal node exists in CLOSED, it checks if there is some goal node in it, such that the upper bound on its cost (for any possible future measurement sequence) is less than the lower bound on the cost of any other goal node, either in CLOSED or below some open node in the graph. When this condition is satisfied, it terminates with that goal node as the decision.

2.3.1. Algorithm B

Let OPEN be the set of open nodes. Let CLOSED be the set of GOAL nodes that have been closed.

Step 0: Set OPEN to the initial state, and CLOSED to null. Set k, the stage variable to 0.

Step 1: Set k = k+1. For each $n \in$ OPEN compute,

$f(n)=g(n)+h(n)+l(n)$, where,

$g(n)=c(N(n))$

$h(n)=0$ if n is a goal node,

$\triangleq \min_{j \in W_n} [\, c(N(j)-c(N(n)) \,]$ if n is not a goal,

$l(n) \triangleq \min_{\bar{X}_k} r(n/Xk,\bar{X}k)$ if n is a goal node,

$\triangleq \min_{j \in W_n} [\min_{\bar{X}_k} r(j/Xk,\bar{X}k)]$ if n is not a goal.

Step 2: Let n* be the node with a minimum value of f(n). If n* is a goal node, remove it from OPEN and put it in CLOSED. If n* is not a goal node, take the measurement associated with that state, and replace n* in OPEN by its successors.

Step 3: For each goal node n in the CLOSED list, compute,

$$b(n) = c(N(n)) + u(n/Xk),$$

where u() is an upper bound on the risk r, i.e.

$$u(n/Xk) \geq \max_{\bar{X}_K} r(n/Xk, \bar{X}k)$$

Let n* be the node(goal) with a minimum value of b(n). If for all n in OPEN $\cup$ CLOSED , $n \neq n^*$,

$$f(n) > b(n^*),$$

STOP, with n* as the B-optimal solution, else go to step 1.

Note that if in step 3 the inequality,

$f(n) > b(n^*)$ is satisfied, the inequality,

$f(n) > f(n^*)$ is also satisfied, because,

$b(n^*) \geq f(n^*)$.

Theorem 5: Algorithm B is B-admissible, i.e. it terminates with the solution, s*, such that,

$c(N(s^*))+r(s^*/X) \leq c(N(s)) + r(s/X)$ for all goals s in G, and X is the total set of features that can be used.

Proof: Algorithm B will terminate when it finds at stage k, a node n* in CLOSED, such that, for all other nodes in OPEN or CLOSED,

$b(n^*) \leq c(N(n))+h(n)+l(n)$.

Suppose n is any node ($\neq n^*$) in CLOSED, then,

$h(n)=0$, $l(n) \leq r(n/X)$, from the definitions of h, l, hence,

$$\begin{aligned} c(N(n^*))+r(n^*/X) &\leq c(N(n^*))+u(n^*/Xk) \\ &= b(n^*) \\ &< c(N(n)) +h(n) +l(n) \end{aligned}$$

$\leq$ c(N(n)) + r(n/X) ,

since h(n) = 0 for any goal node, n.

Hence, n* is a better goal than any n in CLOSED. Suppose n was a goal in Wj for some node j in OPEN. Then,

b(n*) < f(j) = c(N(j)) +h(j) + l(j)

$\leq$ c(N(n)) + r(n/X), because,

by the definition of h , l,

h(j) $\leq$ c(j,n) for any goal n in Wj, where c(j,n) denotes the cost of the path between j and goal n. Also,

l(j) $\leq$ $\min_{X_K}$ r(n/Xk,$\overline{Xk}$) $\leq$ r(n/X) , by the definition of l in step 2 of algorithm B.

Hence,

c(N(n*)) + r(n*/X) < b(n*) < c(N(n)) + r(n/X) .

Thus, n* is a better goal than any goal n, under any open node j. Since all the goal nodes are either in CLOSED or under some OPEN node, and none is better than n*, across all measurements, X, n* is B-optimal, and the algorithm is B-admissible.

In Theorem 4, a method of comparing two S-admissible algorithms was presented. The next theorem allows one to compare two B-admissible algorithms using different bounding functions, u. It states that if two B-admissible strategies use the same evaluation function to order the open nodes, but one uses a more 'informed' upper bounding function than the other, it can never expand more nodes than the less informed strategy.

Theorem 6: Consider two B-admissible strategies using the same functions, h,l, but different upper bounding functions, u, and u', such that, for all goal nodes, n, and measurment sets, Xk,

u(n/Xk) < u'(n/Xk) .

Then the strategy using u′ expands all the nodes expanded by the strategy using u.

Proof: Since both the strategies use the same functions, h, and l, they use the same evaluation function, f, to decide the node to be closed. Hence given that both strategies have gone through k nodes, it follows that both expand the same nodes in the same sequence. Thus, the only question is to decide, which strategy terminates first, i.e. for which strategy is the condition,

$b(n^*) < f(n)$ satisfied earlier.

Since, $c(N(n^*)) + u(n^*/Xk) < c(N(n^*)) + u'(n^*/Xk)$,

for all Xk, it is clear that, the termination criterion in step 3 of the algorithm, will be satisfied earlier for the strategy using u, than for that using u′. Hence u′ will expand all nodes expanded by u.

Unfortunately, one cannot compare two strategies using different heuristics, h+l, and say h′+l′, since in this case, the two strategies would expand different nodes, and it is possible for the less informed strategy, i.e. the one with the smaller value of h+l at each node, to terminate earlier, because the measurement it takes may reduce the bound b(n*) sharply thus causing other classes to get rejected earlier, than for the more informed strategy.

2.3.2. Bayes Optimal Strategy

A Bayes optimal classification scheme is one which minimizes the average risk. Let X be the total set of measurements on a sample, then if there are M classes, w1,w2..wM, the average risk of classifying X in wi, is,

$$r(wi/X) = \sum_{j=1}^{M} cij.\ p(wj/X)$$

where cij is the loss incurred in classifying a sample from wj into wi, and p(wj/X) is the likelihood of wi given X. Hence, a Bayes optimal strategy would classify X into class w* such that

$$r(w^*/X) = \min_{1 \leq i \leq M} r(w_i/X)$$

From the above equation, it follows that such a strategy can be implemented as a special case of a B-admissible strategy, wherein the arc costs do not influence the cost of a goal s*. The only change required in algorithm B is in Step 3 which becomes :

Step 3: For each goal node n in CLOSED compute the upper bound u(n/Xk), on the risk r(n/X), given the measurements, Xk have been observed so far. Let n* be the node with minimum value of u(). Then, if for all nodes, n in OPEN$\cup$ CLOSED, other than n*,

u(n*/Xk) < l(n/Xk) is true,

STOP, with n* as the Bayes optimal classification, else go to Step 1.

2.3.3. Graphical Representation of B-admissible Search

The role played by the evaluation heuristic, h+l, and the upper bounding heuristic, u, can be depicted graphically in 2-space. In this space, each node, open or closed, is represented by its values of f(n), and b(n). Though algorithm B computes b(n) only for goal states n in CLOSED, one could define it for non-goal nodes as,

$$b(n) = \min_{j \in Wn} \left[c(N(j)) + \max_{\bar{X}k} r(j/Xk, \bar{X}k) \right]$$

where j is any goal node below node n, i.e. $j \in Wn$.

Figure 2.1 shows a display of all open and closed nodes in the b-f plane. Since $b(n) \geq f(n)$, all points lie above the 45 degree line, OZ. Node a has the minimum value of f and will be closed next, while node b in CLOSED has the smallest value of b(n), and hence, all nodes with $f(n) \geq \min_{n \in CLOSED} b(n)$ will be rejected from further consideration. The reject region is marked in the figure. Thus, the search is reduced to the domain, R, and the nodes in this region contain the potential B-optimal node, or an ancestor of the B-optimal node. Figure 2.2 shows the situation when the algorithm terminates with the solution, n*. All other nodes now fall in the reject region. The movement of the nodes in the b-f space as more measurements are taken, is shown by small arrows in Fig. 2.1. That they should move in such a direction, follows from the fact that f must increase and b decrease as more measurements are taken.

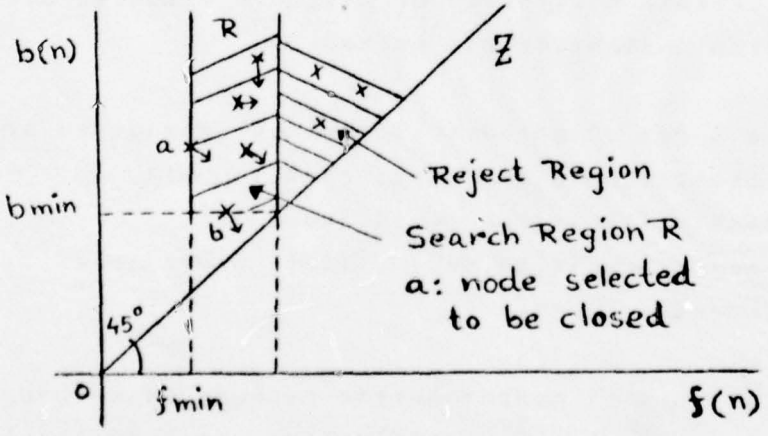


Fig. 2.1

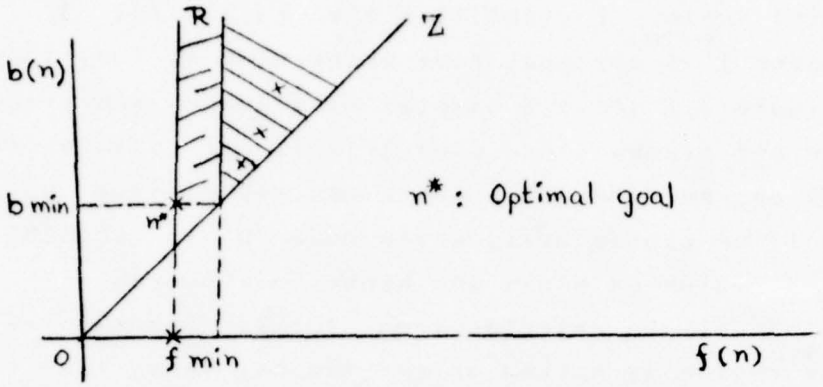


Fig. 2.2

2.4. Methods Of Computing Bounding Functions

Having discussed the properties of S and B-admissible search strategies, we consider here, methods of computing lower and upper bounding functions, l, u, for some special cases. Two broad types of risk functions, r(n) , are considered:

- r is a function of class conditional probability distributions or parameters of distributions (parametric).
- r is defined in terms of a set of labelled samples and certain similarity or distance measures between samples (nonparametric methods) .

The two types of parametric methods considered are:

- the measurements are class conditionally statistically independent, and,
- the measurements satisfy a first order tree dependence.

The two types of nonparametric methods described are:

- nearest neighbour classification using Euclidean distance,

. classification using a similarity measure between binary vectors.

2.4.1. Statistically Independent Features

Assume that the features are class-conditionally statistically independent, i.e. the probability $p(X/wi)$, can be written as,

$$p(X/wi) = \prod_{t=1}^{N} p(xt/wi) \text{ where } X=(x1,x2,\ldots xN) .$$

The average risk , $r(wi/X)$, is given by,

$$r(wi/X) = \sum_{j=1}^{M} cij.P(wj/X) \text{ where } cij \text{ is the}$$

cost of making decision wi, when the correct decision is wj. Assuming a unit loss for incorrect decisions and no gain for correct decisions,

$$r(wi/X) = 1 - \frac{P(wi).\prod_{t=1}^{N} p(xt/wi)}{\sum_{j} P(wj).\prod_{t} p(xt/wj)} \quad \ldots\ldots\ldots(2.4)$$

where $P(wj)$ is the a priori probability of class j.

If $X1 \subset X$ is the set of features already measured, then let $l(wi/X1)$, $u(wi/X1)$, be the lower and upper bounds, respectively, on $r(wi/X)$, i.e.

$$l(wi/X1) \leq \min_{\bar{X}1} r(wi/X1,\bar{X}1)$$

$$u(wi/X1) \geq \max_{\bar{X}1} r(wi/X1,\bar{X}1)$$

where $\bar{X}1$ is the complement of X1 with respect to the total set of measurements, X, upon which r depends.

Define,

$$lt(i) = \min_{\forall x_t} p(xt/wi)$$

$$ut(i) = \max_{\forall x_t} p(xt/wi)$$

Then, from Eqn. (2.4), and the above definitions of lt, ut, we can define the bounds as, follows:

$$l(wi/X1) = 1 - \frac{P(wi).p(X1/wi).\prod_t ut(i)}{\{\ P(wi).p(X1/wi).\prod_t ut(i) + \sum_{j\neq i} P(wj).p(X1/wj).\prod_t lt(j)\ \}} \quad (2.5a)$$

$$u(wi/X1) = 1 - \frac{P(wi).p(X1/wi).\prod_t lt(i)}{\{\ P(wi).p(X1/wi).\prod_t lt(i) + \sum_{j\neq i} P(wj).p(X1/wj).\prod_t ut(j)\ \}} \quad \ldots(2.5b)$$

In equations (2.5a)-(2.5b) above, the product terms in lt, ut, are taken only for those features, xt, that are not in the observations already taken, i.e. $xt \notin X1$.

The above bounds are too pessimistic because they assume that the values of $\bar{X}1$ that minimize/maximize p(X/wi), simultaneously maximize/minimize p(X/wj) for all $j \neq i$ .One could obtain more tight bounds by computing $p(wj/X1,\bar{X}1)$ for all $\bar{X}1$, but for the case of N features, M classes, and each feature taking on m states, this would require $M.m^N$ calculations, which is a large number even for moderately small values of M, m, and N. However, there is a special case, when the tight bounds can be computed with no more computations than are required in cbtaining l, u, using (2.5a-b), above.This is the case where the unconditional distribution of X can be written as a product, i.e.

$$p(X) = \prod_{t=1}^{N} p(xt)$$

Then one can define,

$$lt(i) = \min_{\forall x_t} p(xt/wi) \ / \ p(xt)$$

$$ut(i) = \max_{\forall x_t} p(xt/wi) \ / \ p(xt)$$

Writing p(X) for the denominator of (2.4), and using the above definitions, one can define tight lower and upper bounds on the likelihood of class wi, given observations,

X1, as,

$$l'(wi/X1) = 1 - \frac{P(wi).p(X1/wi).\prod_t ut'(i)}{p(X1)} \qquad (2.6a)$$

$$= 1 - P(wi/X1).\prod_t ut'(i)$$

$$u'(wi/X1) = 1 - P(wi/X1).\prod_t lt'(i) \qquad (2.6b)$$

Using the bounds given by (2.5a-b), or (2.6a-b), requires computing the 2NM quantities, lt, ut, or lt', ut' respectively.

Example :

Consider the following example in which 4 features, x1..x4, are available to classify the sample into one of 3 classes. Each feature can be in one of 4 states, a,b,c and d, and it is assumed that the features are class-conditionally statistically independent. The probabilities are tabulated, and the maximum and minimum values of p(xj/wi) are shown in the last two rows for each class wi, and each feature, xj. These are the values uj(i), lj(i) respectively.

	Class 1				Class 2				Class 3			
	x1	x2	x3	x4	x1	x2	x3	x4	x1	x2	x3	x4
a	.4	.3	.2	.2	.5	.4	.1	.1	.1	.4	.1	.2
b	.4	.5	.4	.2	.2	.1	.1	.1	.3	.1	.3	.3
c	.1	.1	.2	.4	.2	.3	.6	.1	.5	.2	.3	.2
d	.1	.1	.2	.2	.1	.2	.2	.7	.1	.3	.3	.3
ut	.4	.5	.4	.4	.5	.4	.6	.7	.5	.4	.3	.3
lt	.1	.1	.2	.2	.1	.1	.1	.1	.1	.1	.1	.2

Assume that features x1 and x2 have been observed, and that their values are x1=a, and x2 =b . Using equations (2a) and (2b) one can compute the lower bound and upper bound on the risk for each decision, wi, denoted by, l(wi/x1,x2), and u(wi/x1,x2).

$$u(w1/x1,x2) = 1 - \frac{.4.5.l3(1).l4(1)}{[.4.5.l3(1).l4(1) + .5.5.u3(2).u4(2) + .1.1.u3(3).u4(3)\ ]}$$

$$= 1 - .4.5.2.2/(.4.5.2.2 + .5.1.6.7 + .1.1.3.3)$$

= 0.733

Similarly,

$l(w1/x1=a, x2=b) = 1 - 320/327 = 0.0021$

$u(w2/x1=a, x2=b) = 1 - 5/334 = 0.985$

$l(w2/x1=a, x2=b) = 1 - 210/292 = 0.280$

$u(w3/x1=a, x2=b) = 1 - 2/532 = 0.995$

$l(w3/x1=a, x2=b) = 1 - 9/94 = 0.905$

From the above values it is clear that class w3 can be rejected from consideration, since after the first two measurements, the lower bound on the risk of making the decision 'class 3', viz. l(w3/X1), is greater than the upper bound on the risk of deciding in favour of class w1, viz. u(w1/X1) .

2.4.2. Tree Dependent Features

In many applications the assumption of statistical independence is unjustifiable, and one seeks a more elaborate model of feature dependence. One such model is that of first order tree dependence [23], wherein, the class conditional distribution of X is,

$$p(X/wi) = p(x1/wi).\prod_{j=2}^{N} p(xj/xt(j),wi) \text{ where } t(j) < j$$

i.e. the features can be ordered so that the probability of X can be written as a product of first order conditional probabilities. The dependence between the features can be represented by a dependence tree, Td, such that there are nodes labelled x1,x2,...xN, i.e. the feature names, and there is a directed edge from node t(j) to j for j=2,..N. For the tree of Fig. 2.3 for example, p(X/wi) can be written as,

$$p(X/wi) = p(x1/wi).p(x2/x1,wi).p(x3/x1,wi).p(x4/x2,wi).p(x5/x2,wi)$$

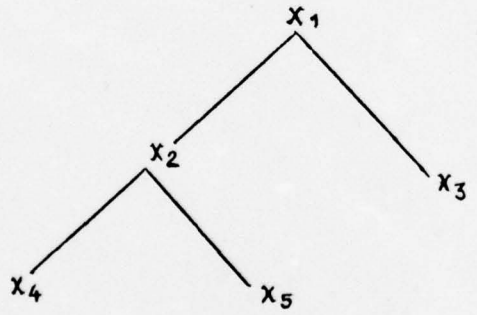


Fig. 2.3

It is assumed in the following discussion that the features, xt, t=1,2..N, are discrete, xt, taking on one of mt states. We shall define a measurement graph, a spanning tree and a real valued function on a tree, and then describe

a method of computing lower/upper bounds on the risk r(wi/X) given that a subset of features $X1 \subset X$, have been observed.

Definition: A measurement graph, G(Td,wi), for class wi, and dependence tree, Td, is defined thus: If x1,...XN are the labels of nodes in Td, i.e. features, then the nodes of G can be divided into disjoint sets, A1,...AN, such that the nodes in Ak are:

$$Ak = \{ xk(1), xk(2), \ldots xk(mk) \}$$

i.e. xk(j), j=1,2,...mk are the states of feature xk. There is a directed edge from xk(j) to xl(m) in G if there is an arc from node xk to xl in Td. The arc in G has a 'cost' given by,

$$c(xk(j),xl(m)) = \log p(xl(m)/xk(j),wi)$$

Thus, the arc costs in G are the log of the conditional probabilities in the product form for p(X/wi).The measurement graph corresponding to Fig. 2.3, assuming binary features, is shown below.

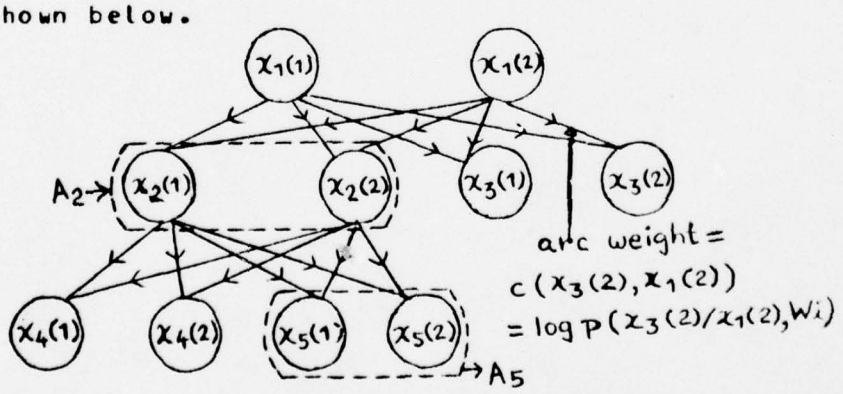


Fig. 2.4 Measurement Graph

Definition: A terminal node of graph G is one having no outgoing edges from it, e.g. in Fig. 2.4, x3(1), x3(2), x4(1), x4(2), x5(1), x5(2) .

Definition: A spanning tree, Ts, of graph G(Td,wi) is defined as one containing exactly one node from each node set, Ak, k=1,2..N in G, and the arcs joining these nodes. This definition is different from the use of the term 'spanning tree' in graph theory. Examples of spanning trees are shown in Fig. 2.5a-2.5b below.

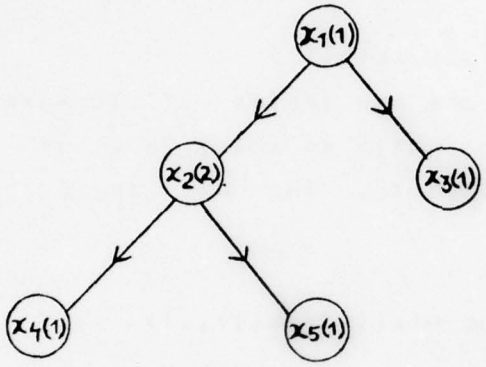


Fig. 2.5a

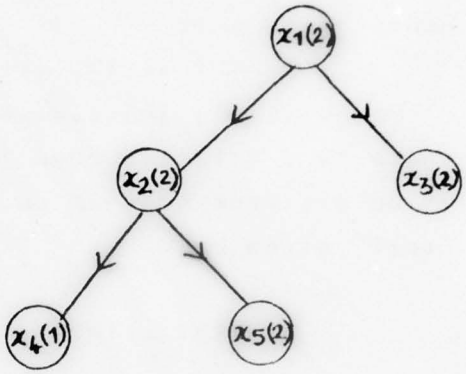


Fig. 2.5b

Definition: The weight of a spanning tree, Ts, is the sum of the arc costs in Ts.

Definition: A minimal/maximal spanning tree is one whose weight is minimum/ maximum among all spanning trees of graph G(Td,wi).

Definition: A constrained spanning tree, Ts(X1) of G is a spanning tree of G containing all the nodes (feature states) in X1. Two constrained trees, Ts(x1(2),x4(1),x5(2)) are shown below.

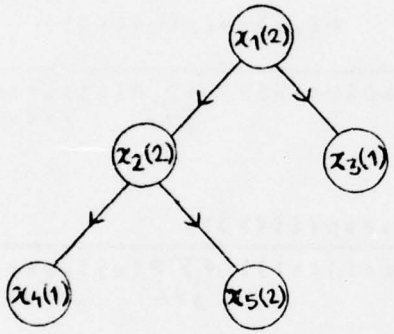


Fig. 2.6a

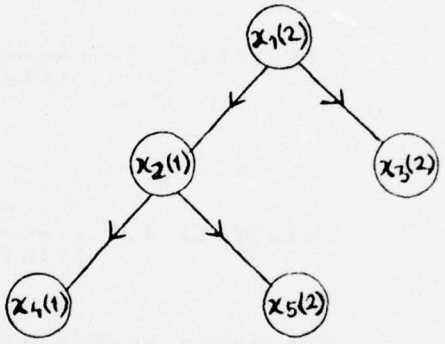


Fig. 2.6b

Definition: A minimal/maximal constrained spanning tree is a constrained tree having minimum/maximum weight among all such trees.

If d(T) denotes the weight of tree, T, then the minimal and maximal weights of Ts(X1), denoted by li(X1), ui(X1), are,

$$li(X1) = \min_{\forall T_s(X_1)} d(Ts(X1)) \qquad \text{.....(2.7a)}$$

$$ui(X1) = \max_{\forall T_s(X_1)} d(Ts(X1)) \qquad \text{.....(2.7b)}$$

Since the arc weights are logs of the conditional

probabilities, li, ui, are the log functions of the minimum/maximum values of p(X/wi) given that measurements, X1 have been taken.

$$li(X1) = \min_{\bar{X}_1} [\ \log p(X1,\bar{X}1/wi)\ ]$$

$$ui(X1) = \max_{\bar{X}_1} [\ \log p(X1,\bar{X}1/wi)\ ]$$

In the above equations $\bar{X}1$ is the measurement set not yet observed. Substituting for p(X/wi) in equ. 2.4, the bounds become,

$$l(wi/X1) = 1 - \frac{P(wi).\exp(ui(X1))}{\{P(wi).\exp(ui(X1)) + \sum_{j\neq i} P(wj).\exp(lj(X1))\}} \quad \ldots\ldots (2.8a)$$

$$u(wi/X1) = 1 - \frac{P(wi).\exp(li(X1))}{\{P(wi).\exp(li(X1)) + \sum_{j\neq i} P(wj).\exp(uj(X1))\}} \quad \ldots\ldots (2.8b)$$

The above bounds are loose because it has been assumed that the measurements $\bar{X}1$ which minimize/maximize p(X/wi) simultaneously maximize/minimize p(X/wj) for all $j \neq i$. Tight bounds can be obtained with no more computation than that required for equation (2.8a)-(2.8b), if it is assumed that the unconditional probability of X, p(X), can also be written as a product of first order conditional probabilities, as,

$$p(X) = p(x1).\prod_{j=2}^{N} p(xj/xt(j)) \quad \text{for } j=2,\ldots.N$$
$$\text{where } t(j) < j \ .$$

In this case, we define an augmented measurement graph as follows:

Definition: An augmented measurement graph, G'(Td,wi)

for class wi, and dependence tree, Td, is identical to the graph G(Td,wi), except that the arc cost, c(xk(j),xl(m)) is,

$$c(xk(j),xl(m)) = \log p(xl(m)/xk(j),wi) - \log p(xl(m)/xk(j))$$

The second term is the log of the probability of xl(m) given xk(j), but not conditioned on wi.

One can then define the min/max weight of a constrained spanning tree of the augmented graph, G'(Td,wi), for a measurement set, X1, as, li'(X1), ui'(X1) respectively, as before. The bounds on the risk of decision wi, given X1, become,

$$l'(wi/X1) = 1 - \max_{\bar{X}_1} \frac{P(wi).p(X1,\bar{X}1/wi)}{p(X1,\bar{X}1)}$$

$$l'(wi/X1) = 1 - P(wi).\exp(ui'(X1)) \quad \ldots\ldots\ldots(2.9a)$$

Similarly,

$$u'(wi/X1) = 1 - P(wi).\exp(li'(X1)) \quad \ldots\ldots\ldots(2.9b)$$

To use the bounds defined by equations (2.8a-b), (2.9a-b), one needs an efficient method of computing the minimal/maximal spanning trees for a graph. One such method, which is a modification of the dynamic programming formulation for finding the shortest path through a graph, is presented next.

2.4.2.1. Computation Of Minimal/Maximal Spanning Tree

Assume that one wants to compute li(X1), ui(X1) ,the minimal/maximal spanning tree weights, for a measurement graph, G(Td,wi). Denote by Ak the set of nodes of G labelled, xk(1), xk(2), ..xk(mk), where mk is the number of

possible states of feature xk. Let DT(xk) be the descendant nodes of node xk in the dependence tree Td, and DG(xk(j)), the descendant nodes (sons) of node xk(j) in G. The following algorithm recursively computes the longest/shortest path length from a node to all terminal nodes in G. For the case when the measurement graph, G, consists of N features, each taking on one of m states, this algorithm requires computations of the order of $2.(N-1).m^2$

Step 1: Prune G, as follows: If $xk(j) \in X1$, the observed set of features, delete all nodes (and attached edges) in Ak from G, except xk(j). In this way for each observed feature, all states except the observed one are deleted.

Step 2: Compute the functions li(xk(s)), ui(xk(s)), for each node xk(s) in the pruned graph using the recursive equations:

$$li(xk(s)) = \sum_{xj \in DT(xk)} \; \left[\min_{xj(t) \in DG(xk(s))} \{ c(xk(s), xj(t)) + li(xj(t)) \} \right]$$

$$ui(xk(s)) = \sum_{xj \in DT(xk)} \; \left[\max_{xj(t) \in DG(xk(s))} \{ c(xk(s), xj(t)) + ui(xj(t)) \} \right]$$

where c(xk(s), xj(t)) denotes the arc cost along the edge joining nodes xk(s), xj(t) in G, and for terminal nodes of G,

$$li(xj(t)) = ui(xj(t)) = 0 \text{ for } xj(t) \text{ terminal .}$$

If x1 is the root node of the dependence tree, Td, then the lower/upper bounds, li(X1), ui(X1) are,

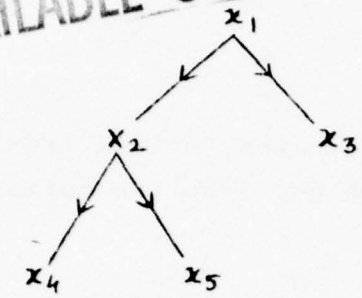
$$li(X1) = \min_{t} li(x1(t))$$

$$u_i(X1) = \max_{t} u_i(x1(t))$$

The correctness of the above algcrithm follows from the additivity of arc costs along a path and from the optimality principle of dynamic programming.

Example:

The following example illustrates the use of the algorithm for finding the minimal/maximal spanning trees of a measurement graph. The example considered is that of Fig. 2.3 assuming that each feature is binary. The first-order conditional probabilities, conditioned on class w1, and the corresponding measurement graph, G(Td,w1) are shown in Fig. 2.7a and 2.7b. Fig. 2.7c shows the minimal and maximal paths traced by the algorithm. The dotted edges indicate the maximal tree, the solid edges, the minimal tree. Arc costs are log(base 10) functions of the probabilities. Next to each node in Fig. 2.7c, the pair of numbers "(a,b)", are the minimum/maximum costs, viz., $l_i(x_k(s))$, and $u_i(x_k(s))$ defined in step 2 of the algorithm. For terminal nodes of the measurement graph, these numbers are both zero. The algorithm backs up from the terminals and computes (a,b) for higher level nodes using the recursive equations described in step 2. The values of the unconstrained minimal and maximal tree weights are -2.962 and -.927 respectively. These correspond to the measurement vectors, (x1=0,x2=1,x3=1,x4=2,x5=1) and (11101) which give rise to the min/max values of p(X/w1), viz.,0.00108 and 0.118 respectively.



Dependence Tree, Td

Fig. 2.7a

$P(x1/w1)$

	x1=0	x1=1
p(x1/w1)	.3	.7

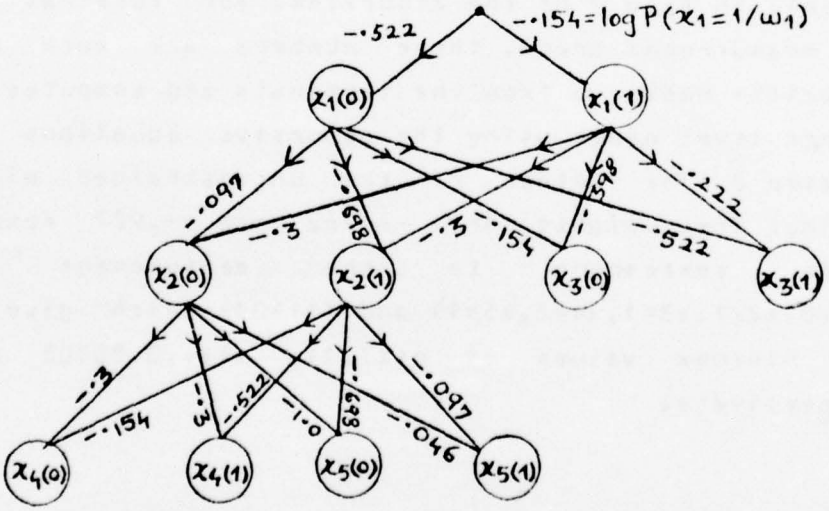
	x1=0	x1=1
x2=0	.8	.5
x2=1	.2	.5

	x1=0	x1=1
x3=0	.7	.4
x3=1	.3	.6

$P(x_2/x_1, w_1)$

	x2=0	x2=1
x4=0	.5	.7
x4=1	.5	.3

	x2=0	x2=1
x5=0	.1	.2
x5=1	.9	.8



Measurement graph, G(Td, w_1)

Fig. 2.7b

Minimal / Maximal Spanning Trees of $G(T_d, \omega_1)$

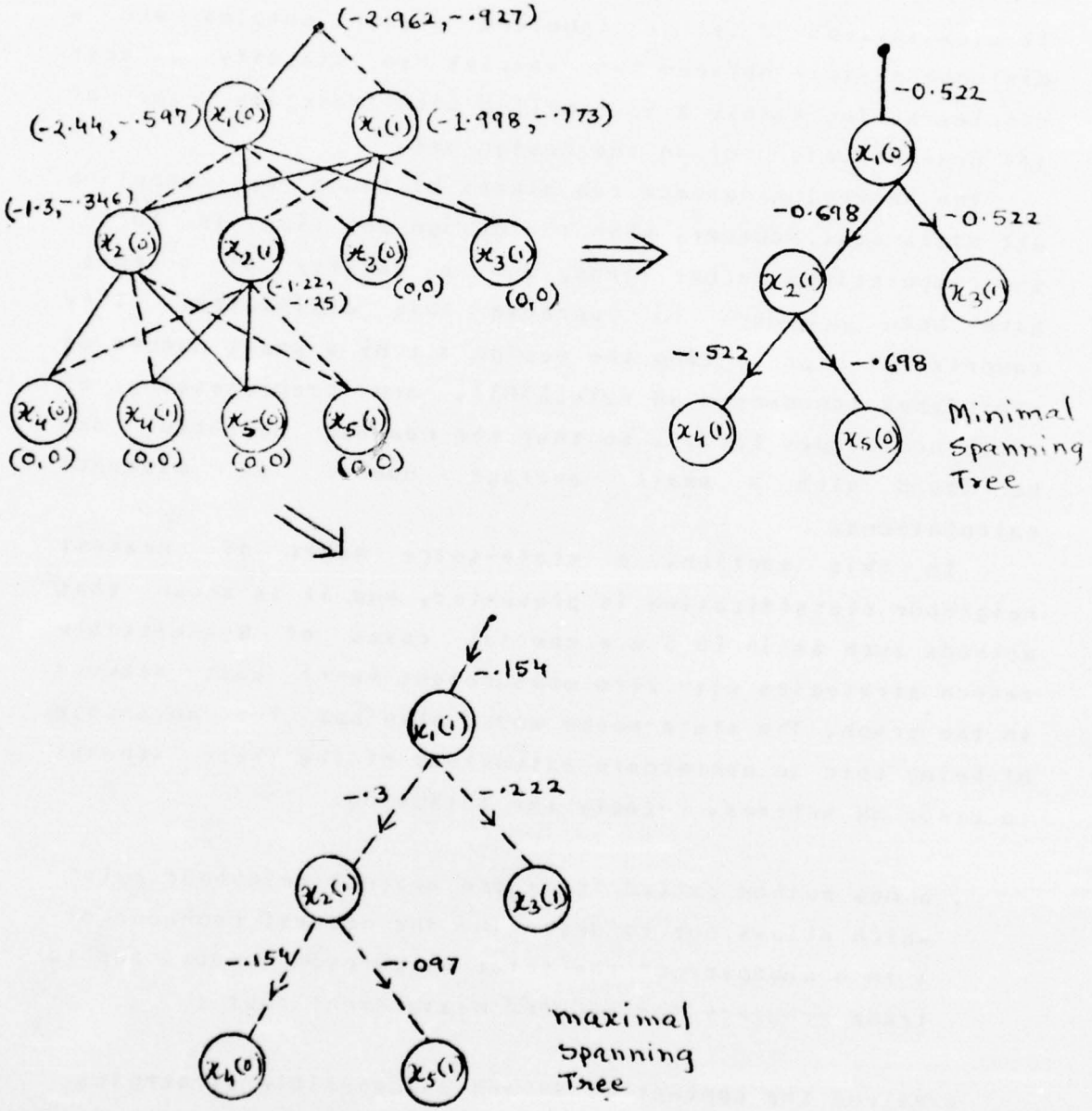


Fig. 2.7c

2.4.3. Nearest Neighbour Classification

A nearest neighbour (NN) classification scheme is a nonparametric method, i.e. it makes no assumptions regarding the form of the class conditional distributions of features. It uses instead, a set of labelled design samples and a distance measure between two samples to classify a test sample, X. The sample X is labelled with the class name of its nearest neighbour in the design set.

The nearest neighbour can always be found by computing all distances. However, when the design set size is large, the computations become large, and a variety of schemes have been proposed to overcome this difficulty. They comprise of representing the design set by a small number of prototypes (condensed NN rule,[30]), and preprocessing of these prototypes [6,31], so that the nearest neighbour can be found with a small average number of distance calculations.

In this section, a state-space model of nearest neighbour classification is presented, and it is shown that methods such as in [6] are special cases of B-admissible search strategies with zero measurement (arc) cost assumed in the graph. The state-space model also has the advantage of being able to accomodate extensions of the these schemes to other NN schemes, notably the following:

- a new method called 'subspace nearest neighbour rule' which allows one to determine the nearest neghbour of X in a subspace of the total measurement space, and to trade accuracy for reduced measurement cost ;

- extend the concept of S- and B-admissible strategies to schemes using non-metric features and a similarity

measure between samples to label a test sample, X. The method of using prototypes to represent the design set in such cases and methods of computing lower/upper bounds on the similarity measure are discussed.

2.4.3.1. The Model

A state space model for non-parametric schemes may be defined as a 7-tuple, G(S,E,X,W,c,r,I), as before, but with the following changes in the connotation of certain symbols:

A state s represents a subset, Ds, of samples from the total design set, D, and is denoted by a 3-tuple, (xs,Ws,Is), where,

xs: is the measurement to be taken on the test sample when this state is reached,

Ws: the classes to which the samples in Ds belong. This set is known since the samples are labelled.

Is: denotes information about the sample set Ds, e.g. its mean vector, measures of its dispersion, etc.

The same sample can occur in two different sets, Dsi, Dsj where states si, sj may not be related (one may not be a descendant of the other). However, one restriction on the sets is that if state t is a descendant of state, s, then $Dt \subset Ds$ must hold.

The real valued function c on the edges E of the state space graph, represents the cost of traversing an arc and could represent either the measurement cost, or the cost of computing the distance measure between the test sample and the samples in that set, Ds. The function r, used in earlier discussions to denote the misclassification risk, is used here to represent the distance between the test sample, X, and a particular design sample, Ys*. A goal state s* is

one for which $|Ds^*| = 1$, i.e. it represents a single sample, say Ys^*. Hence,

$r(s^*) = d(X, Ys^*; F)$, where,

d is the distance measure used ,

F is the space of features in which d is computed,

X, Ys^* are the test and design samples respectively.

Let,

Fs be the features measured on the path to state s,

c(Fs) be the sum of arc costs on the path to s.

Then the following types of nearest neighbour schemes can be formulated :

Scheme 1: Find the goal s^*, i.e. the design sample Ys^*, such that,

$$f(s^*) = c(Fs^*) + d(X, Ys^*; Fs^*)$$
$$= \min_{s} [\, c(Fs) + d(X, Ys; Fs) \,]$$

where s is any goal node.

Scheme 2: Find the goal s^*, such that,

$$f(s^*) = c(Fs^*) + d(X, Ys^*; F)$$
$$= \min_{s} [\, c(Fs) + d(X, Ys; F) \,]$$

where s is any goal node, and,

F is the total set of features.

Scheme 3: Find the goal s^*, such that,

$$f(s^*) = d(X, Ys^*; F) = \min_{s} [d(X, Ys; F)]$$

where s is any goal state.

Schemes (1) and (2) assume that measurement cost (arc cost) is significant, and hence attempt to trade this cost

for reduced accuracy, i.e. it is likely that they may not terminate with the truly nearest neighbour. Scheme (3) is the conventional NN scheme assuming that arc costs do not contribute to the solution cost. Schemes (1) - (3) can be implemented as S-admissible, B-admissible, and Bayes admissible strategies respectively.

Scheme(1) assumes that the design sample, s*, is adequately represented by the features, Fs*, measured on the path to it, and that other features are not significant in the distance measure i.e., d(X,Ys*;F) can be approximated by d(X,Ys*;Fs*). Hence the space in which the d measure is computed depends upon the design sample and its class label. Such an assumption may be justified ,for example, in a character recognition scheme, where features such as the 'horizontal top line' or 'curly tail' may be significant for some class, whereas the slant of the axis of symmetry of the letter may vary with writing styles, and considered unimportant for distinguishing that class.

Scheme (2) does not assume that any features are unimportant for the d measure, but it does associate an arc cost with every edge and hence trades the risk of not finding the true nearest neighbour for reduced feature measurement cost.

The crucial problem in using the admissible search strategies discussed earlier, is that of computing lower/upper bounds on the d() measure. The following sections derive bounds for various metric and non-metric similarity measures.

2.4.3.2. Euclidean Measure

Assume that d(X,Y;F) is the Euclidean distance between X, Y in the space of features, F. Ds is the set of samples represented by any state s, and Is, the information

associated with s. We shall assume that Is consists of the following :

Ms : Mean of samples in Ds, in the total space, F.

Fs : features measured on the path to node s.

Rsmax(F′) : distance between Ms and the sample Y, in Ds farthest from Ms, in the space F′.

Rsmin(F′) : distance of sample closest to Ms in F′.

From the nature of the d measure,

$$d(X,Ys;F) \geq d(X,Ys;F') \quad \text{if } F' \subset F .$$

Lemma :
$$\min_{Y \in D_s} d(X,Y;F) \geq d(X,Ms;F') - Rsmax(F')$$

Hence the right handside of the above inequality provides a lower bound on the nearest neighbour's distance from X, from among the samples in set Ds.

Proof: For any sample $Y \in Ds$,

$$d(X,Y;F') + d(Y,Ms;F') \geq d(X,Ms;F')$$

from the triangular inequality property of d. Hence,

$$d(X,Y;F) \geq d(X,Y;F') \geq d(X,Ms;F') - d(Y,Ms;F')$$
$$\geq d(X,Ms;F') - Rsmax(F')$$

Hence,

$$\min_{Y \in D_s} d(X,Y;F) \geq d(X,Ms;F') - Rsmax(F')$$

If at node s, the information stored is Ms, and Rsmax(Fs), then the lower bound derived above can be used in an S-admissible strategy for Scheme (1) defined earlier. The lemma is a generalization of the bound proposed by Fukunaga [6] for a branch-and-bound algorithm for determining nearest neighbours. Scheme (1) has the advantage that the radius, Rsmax has to be computed in a single dimensioned feature

space for all level 1 nodes in the state-space graph, in two dimensions for level 2 nodes, etc. Thus, at the initial node, where Ds is the total design set, the computation of Rsmax is the easiest, and successively becomes more complex as the number of samples decrease. In the scheme described in [6], the radius Rsmax is always computed in the total feature space, F, and hence, far more computations are involved. Of course, scheme (1) is valid only where a subspace representation of each design sample is justified.

Lemma : $\min_{Y \in D_s} d(X,Y;F) \leq d(X,Ms;F) + Rsmin(F)$

Hence the right handside is an upper bound on the distance of the nearest neighbour in Ds from X.

Proof : Let Y' be the point nearest to Ms in F, then,

$$d(X,Y';F) < d(X,Ms;F) + Rsmin(F)$$

Hence,

$$\min_{Y \in D_s} d(X,Y;F) \leq d(X,Ms;F) + Rsmin(F)$$

The use of this upper bound involves computing d in the total feature space, F, and all features of X must be measured. Thus this is not suitable for Scheme (2) where we would like to compute the upper bound without incurring this measurement cost. However, the bound can be used in scheme (3) where measurement cost is assumed to be zero. Without some other knowledge about the range of feature values, it is not possible to upper bound the Euclidean distance for use in scheme (3) .

2.4.3.3. Ultrametric Measure

An ultrametric satisfies the property,

$d(X,Y) \leq \text{Max}\ [\ d(X,Z),\ d(Z,Y)\ ]$ for any X, Y, Z,

and hence also satisfies the triangular inequality. Hence the same lower bound as defined for the Euclidean measure can be used, and if Ysp is any prototype sample in Ds, the distance, d(X,Ysp) can be used as an upper bound on the minimum distance between X and any sample in Ds.

2.4.4. Bounds on Similarity Measures For Binary Vectors

Nearest neighbour schemes can be used for samples with non-metric features if appropriate distance or similarity measures are defined between samples. If a similarity criterion is used, a sample X is classified in the class to which the sample (design) most similar to it belongs. This section considers the possibility of representing subsets of the design samples say, Ds, at each state s, in a state space graph, by 'prototypes'. The prctotypes at s are said to 'cover' the set Ds, if given the prototypes, one can generate a set D' such that $Ds \subset D'$. Thus the prototypes play a role similar to the mean Ms, and radii, Rsmax, Rsmin, discussed in the nearest neighbour schemes using the Euclidean distance. Once such prototypes are generated, one would like to bound the value of the similarity measure, say S(X,Y) between X and any $Y \in Ds$, by using the information in the prototypes.

Consider the case when the samples are N bit binary vectors. A prototype vector, t will be assumed to be an N-bit vector whose bits can have values, '0', '1', or '-' (don't care) . The set D(t) is the set obtained by replacing the '-' bits in t by all possible permutations of '0's and '1' s. Thus, if, t=(-0-10), then D(t) ={00010,00110,10010,10110} . Methods of generating good prototypes for the design samples from each class for the discrete feature case, are discussed in [13,15].

The model for NN search using S(X,Y), a similarity

measure between binary vectors, is identical to that for the nearest neighbour schemes discussed earlier. The task here, is to maximize the value of S(X,Y). The information Is at node s is assumed to be a set of prototypes, {ts(i)} , i=1,2,...ps, such that,

$$Ds \subset D(ts(1)) \cup D(ts(2)) \cup \ldots\ldots \cup D(ts(ps))$$

Define

$$Sl(X,t) = \min_{Y \in D(t)} S(X,Y) \quad \text{and} \quad Su(X,t) = \max_{Y \in D(t)} S(X,Y)$$

Then, it follows that,

$$\min_{Y \in D_S} S(X,Y) \geq \min_{1 \leq i \leq P_S} [\, Sl(X,ts(i)) \,]$$

$$\max_{Y \in D_S} S(X,Y) \leq \max_{1 \leq i \leq P_S} [\, Su(X,ts(i)) \,]$$

The bounds, Sl, Su, for various types of measures, S, are derived in the next section.

2.4.4.1. Bounds, Sl, and Su

The following notation is used in the discussion in this section.

X, Y :test sample and a design sample respectively.
Both are binary vectors.
N : the number of bits in X, Y.
S(X,Y): similarity criterion used.
t : a prototype vector whose bits can be 0,1 or "-".
u(t) : the number of "-" bits in t.
n1(t): the number of "1" bits in t.
n1(X): the number of "1" bits in any binary vector, X.
m(X,Y): the number of matching bits in vectors X, Y.
m(X,t): number of matching bits in X and prototype t,

where only matches in positions at which t has 0/1 bits are considered.

n1(X,-): number of '1's in X in positions where t has a '-' bit.

The following lemmas derive bounds Sl, Su in terms of the above defined quantities for three types of similarity measures commonly used[1].

Lemma : Let $S(X,Y) = m(X,Y) / N$

Then, $Sl(X,t) = \min_{Y \in D(t)} S(X,Y) = m(X,t) / N$

and $Su(X,t) = \max_{Y \in D(t)} S(X,Y) = \{m(X,t) + u(t)\} / N$

Proof: The proof follows from the fact that S can be minimized (maximized) by choosing the '-' bits in t to be opposite (the same as) the corresponding bits in X.

Lemma : Let $S(X,Y) = \dfrac{N \cdot m(X,Y)}{n1(X) \cdot n1(Y)}$

Then,

$$Sl(X,t) = \min\left[\frac{N \cdot \{m(X,t) + n1(X,-)\}}{n1(X) \cdot \{n1(t)+u(t)\}} \;,\; \frac{N \cdot m(X,t)}{n1(X) \cdot \{n1(t)+u(t) - n1(X,-)\}}\right]$$

$$Su(X,t) = \max\left[\frac{N \cdot \{m(X,t) + u(t) - n1(X,-)\}}{n1(X) \cdot n1(t)} \;,\; \frac{N \cdot \{m(X,t) + u(t)\}}{n1(X) \cdot \{n1(t) + n1(X,-)\}}\right]$$

Proof: Let Y be any vector in D(t), and let,

x = # of '1's in Y in the '-' positions of t,

y = # of matching bits between X, Y in the '-' positions of t.

Then for this vector Y,

$$S(X,Y) \triangleq \frac{N.m(X,Y)}{n1(X).n1(Y)} = \frac{N.[\,m(X,t) + y]}{n1(X).[n1(t)+x]} \quad .....(2.10)$$

Equation (2.10) shows that for a given value of x, i.e. the number of '1's in Y at the '-' pcsitions in t, y can be minimized, and hence, S, by assigning as many of these ones to positions at which X has zeroes, thus causing maximum mismatch between X, Y. In particular, y=0 if x= u(t)-n1(X,-), and the ones in Y are assigned in this manner. Decreasing x below this value would cause y to increase; hence the minimum value of S must be attained in the x range,

$$u(t) - n1(X,-) \leq x \leq u(t)$$

$$\text{and} \quad y = x-[\,u(t) - n1(X,-)\,] \quad .....(2.11)$$

Substituting for y in (2.10) using (2.11) and minimizing over the specified range of x gives the desired result for Sl.

For maximizing S(X,Y), it is noted that for a given number of matches y, x will be minimized, and S maximized, by making the matches occur at the '0's in X, as much as possible.For y= u(t) - n1(X,-), x will be zero if Y has all '0's in the '-' positions in t.If y is decreased below this value, x can only increase. Hence, the range of y is,

$$u(t) - n1(X,-) \leq y \leq u(t) \text{ and,}$$
$$x = y-[\,u(t) - n1(X,-)\,] \quad .....(2.12)$$

Substituting for x in (2.10) using (2.12) and maximizing over the above defined y range, gives the desired value of Su.

Example 1:

Consider a sample, X, and prototype given by,

X = (1011010010)

t = (-----01010)

Here N=10 u(t)=5 n1(X)=5 n1(t)=2 m(X t)=3 n1(X,-)=3.

$$\min_{Y \in D(t)} S(X,Y) = \min\{10.(3+3)/5.(2+5) \ , \ 10.3/5.(2+5-3)\} = 1.5$$

$$\max_{Y \in D(t)} S(X,Y) = \max\{10.(3+5-3)/5.2 \ , \ 10.(3+5)/5.(2+3)\} = 5.0$$

Lemma : Let $S(X,Y) = m(X,Y) \ / \ [\ n1(X)+n1(Y)-m(X,Y)\ ]$

Then,

$$Sl(X,t) = \frac{m(X,t)}{n1(X) + n1(t) - m(X,t) - n1(X,-) + u(t)}$$

$$Su(X,t) = \frac{m(X,t) + u(t)}{n1(X) + n1(t) - m(X,t) + n1(X,-) - u(t)}$$

Proof: Defining x, y as before, S(X,Y) becomes,

$$S(X,Y) = \frac{m(X,t) + y}{n1(X) + n1(t) - m(X,t) + x - y} \qquad (2.13)$$

From (2.13), it is observed that S will be minimized for a given x by minimizing y, and this can be done by assigning the '1's in Y to positions where X has '0' s. A similar argument can be applied for maximizing S. The ranges for x, y and their relationship are identical to (2.11) and (2.12) derived in the previous proof. By minimizing/ maximizing S over these ranges, one gets the desired result.

Example 2: Using the same example as in Example 1,

X= (1011010010)

t= (-----01010)

$$\min_{Y \in D(t)} S(X,Y) = 3 / (5+2-3-3+5) = 0.5$$

$$\max_{Y \in D(t)} S(X,Y) = 3+5 / (5+2-3+3-5) = 4.0$$

The above derived bounds show that for many nontrivial parametric and nonparametric schemes, one can obtain bounds relatively inexpensively, and thereby cut down the measurement cost or the number of distance computations (for nonparametric) needed to classify a test sample. The generality of the model has allowed us to describe new types of nearest-neighbour schemes which use distance measures in subspaces of the total feature space to find the design sample 'most similar' to the test sample. The next section describes a particular type of state-space graph, viz. the hierarchical classifier, as a useful model that restricts the generality of the state-space mocel and thereby improves the search efficiency for practical multiclass recognition problems.

2.5. Hierarchical Classifiers

A state space model is a very general representation of any multistage scheme in that one could have a model, where at any node tk, which is k nodes away from the initial node, one could observe any of the N-k features not observed (assuming N features are available, and at each node, one observes one feature); alternatively, one could classify the sample into any of the M classes. The figure below shows the possible successor nodes of tk.

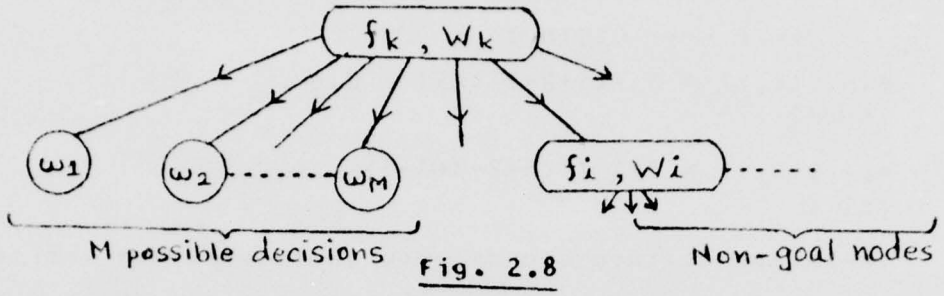


Fig. 2.8

In the above diagram, each state which is not a goal state, is shown as a tuple, (fi,Wi). Since fi can be any of N-k features that have not been observed so far, and Wi is any subset of the M classes, the number of such non-goal states is $(N-k).2^M$. In addition, one has the M possible goal states, hence the total number of successors of a node k levels deep in the graph, is:

$$M + (N - k).2^M$$

While one could still define admissible search strategies for such a general graph, the search efficiency, defined as the average number of nodes expanded(measurements taken) to classify a sample, would be poor.In a practical design, therefore, one might want to use prior information, such as the usefulness of certain features in discriminating subsets of classses, to limit the graph. The graph could be restricted either by restricting the possible successor nodes of a node, or by limiting the states in some way.

A hierarchical classifer is a particular kind of model, wherein the states and possible transitions (edges) in the graph are explicitly defined. The restriction put on the states is that if (fk,Wk) denotes state k, and if state t is a successor of state k, then,

$$Wt \subset Wk$$

This condition implies that the set of possible class labels considered when t is reached, is a proper subset of those considered possible at any of its ancestor nodes. Hence, the decision process can be represented as a tree, at each node of which the set of possible class labels is partitioned into subsets (perhaps overlapping) of labels. Thus along any path in the tree, at each node, at least one class is being 'rejected' from consideration. The term 'rejected' is being qualified because it is quite likely that an admissible strategy might expand nodes on some path and then back up to expand some node (in the OPEN list) on some other path. All the properties of S-admissible and B-admissible strategies can be made use of in searching this tree of decisions. An example of such a tree is shown below, and all the states are indicated explicitly.

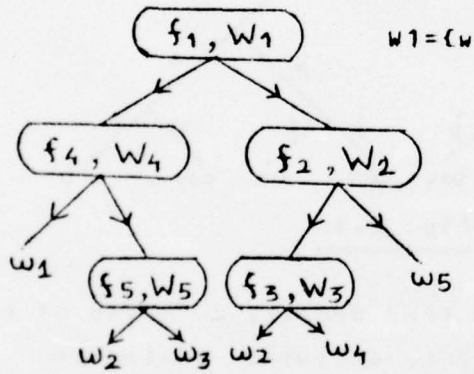


Fig. 2.9

The use of lower and upper bounds in formulating search strategies has been explored in this chapter. However, when the state space graph is small and explicitly defined, as in the above figure, one can obtain a 'perfect' heuristic .In graph searching problems, a perfect heuristic is one which can estimate the path length from any node to the goal node

exactly. Thus to follow the shortest path in the graph, one simply starts at the initial node, computes the heuristic function for all its successors, and goes to that node whose heuristic, added to the cost of that arc (between the current node and it) is a minimum. This concept can be extended to graphs where the goal has a risk which is a function of random variables with a known distribution.

2.6. Hart's Probabilistic Decision Tree Model

The state-space model analyzed in this chapter is closest in spirit to the probabilistic decision tree of Hart[18]. Hart describes a Bayes-admissible search strategy for a decision tree such as shown in Fig. 2.10 below.

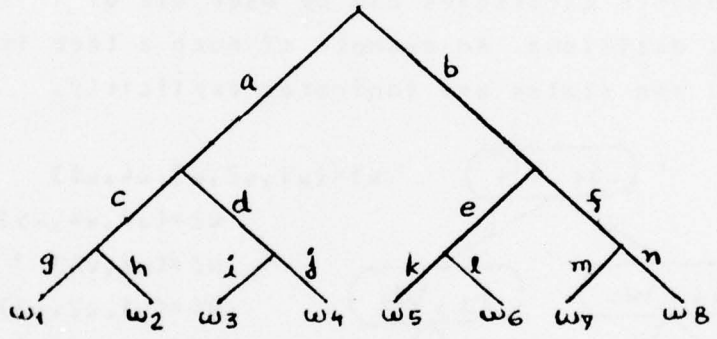


Fig. 2.10

Each edge in the binary tree denotes a 'state of nature' and the terminals represent a joint state of nature comprising of the states on the path to that terminal from the root. Thus, w1={acg}, w2={ach}, etc. Let X denote the total set of measurements taken on the sample, and Xi, i=1,2..n, the measurements taken on the path through nodes 1,2,..n, assumed to lead to node n (say).Let $\bar{\theta}n$ be the joint state of nature comprised of {θ1,θ2,..θn) which are the edges along this path. Then the a posteriori probability that one of the joint states below node n has occurred given

the measurement vector, X, is,

$$f(n) = p(\bar{\theta}n/X) .$$

Hart makes three assumptions regarding X and θn, viz.,

(a) ancestor dependence, i.e. $p(\bar{\theta}n/X)$ depends only on the measurements, X1..Xn on the path to node n. Hence,

$$p(\bar{\theta}n/X) \quad p(\bar{\theta}n/X1,X2..Xn) .$$

(b) conditional independence, i.e.

$$p(X1,..Xn/\bar{\theta}n) = \prod_{i=1}^{n} p(Xi/\theta i) .$$

(c) independence, i.e.

$$p(\bar{\theta}n) = \prod_{i=1}^{n} p(\theta i) \quad \text{and} \quad p(X1,...Xn) = \prod_{i=1}^{n} p(Xi) .$$

Under these conditions, the function f(n) which is used by Hart to order the OPEN nodes, can be written as a product,

$$f(n) = p(\bar{\theta}n/X) = \prod_{i=1}^{n} p(\theta i/Xi) .$$

From the property that f(n) is non-decreasing on any path in the tree and along the sequence of nodes expanded by his algorithm, he proves that the algorithm is Bayes-admissible.

In relation to the three types of strategies discussed in this chapter, it follows that Hart's strategy falls in the category of an S-admissible strategy. Under the assumptions (a)-(c), the function f(n) is an upper bound on the a posteriori probability of any class under n. We have proved the admissibility of such strategies under more general conditions in Theorem 2. In particular, we consider the case where measurement cost is an important factor (Hart assumes zero costs). Moreover, we have shown that for more

general models of feature dependence such as the tree dependence, one can derive lower bounds on the risk for use in an S-admissible strategy.

Hart states that in general cases(i.e. without making assumptions such as (a)-(c) above), it may be difficult to find Bayes-admissible strategies that do not expand the entire search tree. This conjecture has been disproved in our work where we have shown that a B-admissible strategy can be defined for very general situations (viz. where the risk depends on ALL measurements, not just those on the path to a particular goal). The feasibility of computing bounds for use in such strategies has been demonstrated for many parametric and non-parametric schemes.

3. Properties Of Hierarchical Classifiers

This chapter investigates some of the theoretical properties of hierarchical classifiers. Various measures of tree performance are discussed. Emphasis in this chapter, is placed on trees whose node decisions are class conditionally statistically independent. When this assumption is invalid, one can estimate upper bounds on the error made, in terms of the total tree error for any class, and the error of any node classifier.

Two 'negative' properties of hierarchical classifiers make the optimal design problem complex even in those cases where the independence assumption holds. The first property states that in such cases, optimizing each node's performance (Bayes risk) does not minimize the total tree risk. The second property states that choosing at each node of a given tree structure, that feature which makes the node decision with the least average error, is not necessarily the optimal assignment of features. The conclusion is that even for statistically independent node decisions, one cannot optimize the tree performance by optimizing individually, the performance of each classifier used at the tree nodes.

The following notation will be used in subsequent discussions of trees. Consider the tree shown below in Fig. 3.1.

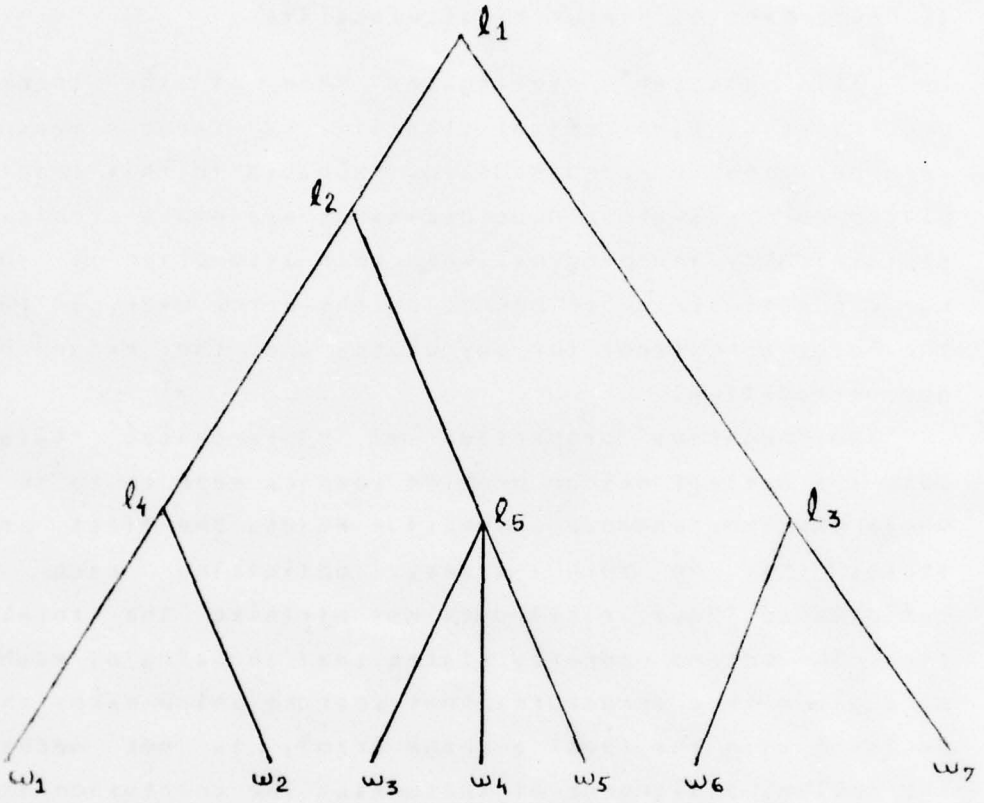

ℓ_1
ℓ_2
ℓ_4
ℓ_5
ℓ_3
ω_1
ω_2
ω_3
ω_4
ω_5
ω_6
ω_7

Fig. 3.1

3.1. Notation

l1,l2.. are nonterminal node labels, l1 is the root.

w1,w2.. are terminal node labels called classes and refer to the type of classification performed when the decision path leads to that node.

M = total number of classes

F = the total set of features

N = the number of features in the set F

fi = the i th. feature name, $i \in \{1,2,...N\}$ for example, 'colour'.

mi = number of states of fi (if discrete)

xi = a random variable, representing an observation of feature fi, e.g. 'red'.

flj = feature (name) used at node lj

mlj = states of feature(if discrete) used at lj.

xlj = random observation of feature used at node lj.

p(x1,x2,..xn/wi) probability of observing values x1,..xn of features, f1,f2,..fn, if sample is from wi.

P(wi) apriori class probability of class wi.

W(lk) set of terminal class labels below node lk, e.g. in Fig. 3.1, W(l2)={w1,w2,w3,w4,w5}.

For a binary tree,

W0(lk), W1(lk) are the terminals of the left and right subtrees below lk, e.g. W0(l1)={w1,w2,w3,w4,w5}, W2(l1)={w6,w7}

S(wi) set of node labels on a path to class wi from the root, e.g. S(w4)={l1,l2,l5}

Pc(wi/lk) probability that a correct decision (e.g. to go 'left' or 'right' in a binary tree) will be made on a random sample from class wi which arrives at node lk. In general, Pc(wi/lk) is a function of the features used along the path to that node

and decision strategy at all previously traversed nodes.

A decision tree,T is composed of a root node l1, a set of nonterminal nodes, {li}, and terminal nodes labelled with the class labels {wj}. The same class label may occur at more than one terminal node. The sample to be classified undergoes a sequence of tests(decisicn rules) on the path from the root to a terminal node, at which point it gets classified as belonging to that category. The choice of the next test to be performed (next node traversed) depends upon the last test done, or in general, upon all measurements taken on the sample.

3.2. Performance Of A Decision Tree

3.2.1. Probability Of Correct Recognition

Consider a tree such as the one in Fig. 3.1. If the features used at the nodes are statistically independent and if at each node, the decision is a function of only that particular feature observation, then the probability of correct recognition of some class, wi, is the product of the correct recognition probability of that class at all nodes on the path leading to that terminal node, i.e.

$$Pc(wi) = \prod_{lk} Pc(wi/lk) \quad \text{where } lk \in S(wi)$$

Hence, the total tree performance, is given by the class recognition rates, weighted by their apriori probabilities, i.e.

$$Pc(T)=\sum_{i} P(wi).\prod_{lk} Pc(wi/lk) \quad i\in\{1,2,\ldots C\},\ lk\in S(wi) \qquad ..(3.1)$$

The average correct recognition rate at a node, lk, for all samples from the set of classes that lie below it, W(lk), is,

$$Pc(lk)= \left[\sum_{i} P(wi).Pc(wi/lk)\right] / \sum_{i} P(wi) \qquad ....(3.2)$$

where $wi\in W(lk)$.

Thus, Pc(lk) is a linear function of the class recognition rates, Pc(wi/lk), whereas Pc(T) involves products of these rates. The design problem is to find a tree, T' such that,

$$Pc(T')= \max_{\forall T} Pc(T)$$

3.2.2. Other Measures Of Tree Performance

While the probability of error is a good measure of tree performance, it is often difficult to compute for arbitrary feature distributions. For two class cases, several bounds on the Bayes error have been suggested [8]. In this section, we consider the use of such measures (bounds) to bound the performance of a decision tree. Let d(wi,wj;F) be some measure of separability of classes wi, wj using features F.If wi, wj, are among a set of M classes distinguished by a tree, T, we wish to define the separability of the classes as a function of the tree. Let lk denote the node at which the two classes get separated, and let $Fk\subset F$ denote the set of features used at that node. If a binary tree is assumed, one could define the separation of wi, wj by T, in any of

the following ways.

$$D(w_i,w_j;T) = d(w_i,w_j;F_k) \qquad \text{.....(3.3a)}$$

$$D(w_i,w_j;T) = \min_{w_s,w_t} d(w_s,w_t;F_k) \qquad \text{.....(3.3b)}$$
where $w_t \in W0(l_k)$, $w_s \in W1(l_k)$.

$$D(w_i,w_j;T) = \sum_s \sum_t C_{st}\, d(w_s,w_t;F_k) \qquad \text{.....(3.3c)}$$
where C_{st} are weights and $w_s \in W0(l_k)$, $w_t \in W1(l_k)$.

$W_0(l_k)$, $W_1(l_k)$, are the sets of classes distinguished at l_k.

The total tree performance, $P_c(T)$ could be defined as,

$$P_c(T) = \sum_i \sum_j D(w_i,w_j;T) \qquad \text{.....(3.4)}$$

3.3. Error in Assuming Sum of Products Form of $P_c(T)$

It has been assumed in the previous discussions that the total performance of a tree, $P_c(T)$ can be written as a sum of products, as,

$$P_c(T) = \sum_{\omega_i} P(w_i) \cdot \prod_{l_k \in S(\omega_i)} P_c(w_i/l_k) \qquad \text{.....(3.5)}$$

Equation (3.5) is exact when the features used at different nodes are class-conditionally statistically independent. Thus, statistical independence is a sufficient though not necessary condition for equation (3.5) to be exactly equal to the true tree performance. In this section, a more general set of conditions is derived for (3.5) to be exact. Also in those cases when these conditions do not hold, an upper bound on the error (in assuming the

product form) is derived in terms of two bounds:

- an upper bound on the error rate (true) of any class by the tree,
- an upper bound on the error rate of any rule used at a node for a sample from some class, wi .

Equation (3.5) is not accurate because the quantity Pc(wi/lk) is the performance on a sample from wi distributed according to the distribution, say, p(X/wi) in nature. However, when this rule is used at the k th. node in a tree, the distribution of the class wi samples that arrive at lk is NOT p(X/wi), in general, but some other distribution, say, p'(X/wi). This is due to the samples that are directed away from the path to wi by the rules used above node lk in the tree. Let Pc(T) be the tree performance, Pc(wi/T) the correct recognition rate for class wi samples using the tree, T, and $\hat{P}c(T)$, the product sum on the right handside of Eqn. (3.5). Then one seeks a bound on,

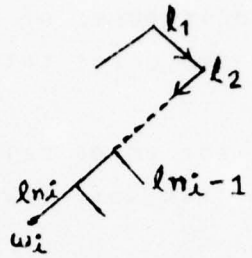
$$|Pc(T) - \hat{P}c(T)| \quad \ldots\ldots\ldots\ldots\ldots\ldots (3.6)$$

where, $Pc(T) = \sum_{\omega_i} P(wi).Pc(wi/T)$

The bound (3.6) may be derived by first bounding the quantity,

$$|Pc(wi/T) - \prod_{l_k \in S(\omega_i)} Pc(wi/lk)| \quad \ldots\ldots\ldots (3.7)$$

For convenience, let l1, l2, ..lni be the nodes on the path to wi in the tree, as shown below.



The total population of wi can be divided into two disjoint

populations, viz. those that trickle down to node lk, and those that do not. Define,

pci(lk) = correct recognition rate of rule at lk on that population of wi samples that reach lk in the tree,
pri(lk) = correct recognition rate of rule at lk if it were used on the samples that do NOT reach lk .
Pi(k-1) = fraction of the total population of class wi that reaches lk via l1, l2, ...lk-1 .

Then, since the two populations defined earlier are disjoint,

$$Pc(wi/lk) = pci(lk).\ Pi(k-1) + pri(lk).\ (1-Pi(k-1)) \quad ..(3.8)$$

Also, the Pi(k-1) are related as follows:

Pi(1) = Pc(wi/l1)
Pi(2) = pci(l2).Pi(1)
Pi(3) = pci(l3).Pi(2) etc.

Hence by back substitution, one cbtains,

$$Pi(ni) = Pc(wi/T) = \prod_{k=1}^{ni} pci(lk)\ l1)/l1).....(3.9)$$

where pci(l1) = Pc(wi/l1) .

Substituting for pci(lk) from equation (3.8) into (3.9),

$$Pc(wi/T) = \prod_{K=1}^{n_i} Pc(wi/lk)\left[1 - \frac{pri(lk).(1-Pi(k-1))/Pc(wi/lk)}{Pi(k-1)}\right]$$

We can then define the ratio of the true and estimated values of Pc(wi/T) as,

$$Ri = \frac{Pc(wi/T)}{\prod_k Pc(wi/lk)} = \prod_{k=1}^{n_i}\left[1 - \frac{pri(lk)(1-Pi(k-1))/Pc(wi/lk)}{Pi(k-1)}\right] \qquad (3.10)$$

From equation (3.10) it follows that Ri =1, i.e. the product form for Pc(wi/T) will be exact ,if,

$$pri(lk) = Pc(wi/lk)$$
$$= pci(lk) \quad \text{for } k=1,2,3....ni \quad .....(3.11)$$

The above result states that if the average performance of the rule at lk on the samples rejected prior to reaching lk, is the same as that on the samples reaching lk, then the product form will be exact. Moreover, the extreme values of Ri occur for pri=1. and pri=0. If pri=0.0 is true, then the product form will be pessimistic, i.e.give too low a value of Pc(wi/T), while if it is 1 , it will be optimistic, i.e. will result in too large a value of the estimated Pc(wi/T). Thus Ri(max) is greater than 1. and Ri(min) is less than 1.

$$Pc(wi/T)\ /Ri(max) < \prod_{k=1}^{n_i} Pc(wi/lk) < Pc(wi/T)\ /Ri(min)$$

Hence,

$$Pc(wi/T) - Pc(wi/lk) < Pc(wi/T)\{\ 1/Ri(min) - 1/Ri(max))$$

$$= Pc(wi/T)\left[\frac{\prod_k Pi(k-1)}{\prod_k\left\{1-\frac{(1-Pi(k-1))}{Pc(wi/lk)}\right\}} - \prod_k Pi(k-1)\right]$$

.... (3.12)

To simplify the above expression, we shall put bounds on the error rate for each class by the tree, and by any particular node. Let,

$(1 - Pc(wi/lk)) < el(i)$

$(1 - Pc(wi/T)) < e(i)$

Then, $1-Pi(k-1) < e(i)$, and hence, equation (3.12) can be written as,

$$\left|Pc(wi/T) - \prod_{k=1}^{n_i} Pc(wi/lk)\right| < Pc(wi/T)\left[\left\{1 - e(i)/(1-el(i))\right\}^{-n_i} - 1\right]$$

$$= Pc(wi/T) \cdot ni \cdot e(i)/(1-el(i)) \quad \ldots\ldots\ldots\ldots(3.13)$$

if $e(i)/(1-el(i))$ is small in comparison to 1 .

Substituting in equation (3.6) using (3.13), one gets,

$$\left|Pc(T) - \hat{Pc}(T)\right| < \sum_{\omega_i} P(wi).Pc(wi/T).ni.e(i)/(1-el(i)) \quad \ldots\ldots\ldots\ldots(3.14)$$

Consider the special case of a balanced tree with n levels in which the upper bounds, e(i) and el(i) are the same for all classes. Then the percentage error in estimating Pc(T) as the product sum on the right handside of equation (3.5), is given by,

$$\frac{\left|Pc(T) - \hat{Pc}(T)\right|}{Pc(T)} < \frac{n \cdot Et}{1 - El} \quad \ldots\ldots\ldots(3.15)$$

In (3.15) , Et is the upper bound on the error rate for any class by the total tree, El is the upper bound on the error rate for any class by any SINGLE node's decision rule, and n is the number of levels in the tree. Equ. (3.15) shows that the error in the estimate increases linearly with the levels in the tree and the class error rate.

3.4. A Property Of The Optimal Decision Policy

In a general decision scheme, the decision at a node, lk, regarding the next node to be traversed, is a function of all the measurements taken on the path to lk from the root. Consider as a special case, those decision policies which depend only on the last feature measured (abbreviated as OS, for one-step policies). Equ. (3.2) provides a means of choosing a policy which maximizes the average correct recognition rate at a node, Pc(lk). This optimal rule is called a 'one-step maximum-likelihood' rule (abbreviated OSMLR), and given by,

OSMLR: If the measured feature value at lk is xlk=s,
if,

$$\sum_{\omega_i \in W_0(\ell_k)} P(w_i).p(x_{lk}=s/w_i) > \sum_{\omega_j \in W_1(\ell_k)} P(w_j).p(x_{lk}=s/w_j)$$

then go left,
else go right.

In the above formulation, a binary decision is assumed at lk, but a similar maximum likelihood rule may be written for the case when lk has m (>2) descendant nodes. The property described below shows that using an OSMLR rule at each node does not necessarily result in the optimal tree performance, Pc(T).

Property 1:

Using a one-step maximum likelihood rule(OSMLR) at each node of a tree,T, does NOT necessarily result in the optimal tree performance, Pc(T), across all possible choices of one-step (OS) rules. This property holds even when the features are statistically independent.

To see why this property holds, consider the binary tree of fig.3.2, in which feature fi, is assumed to be measured at node li.

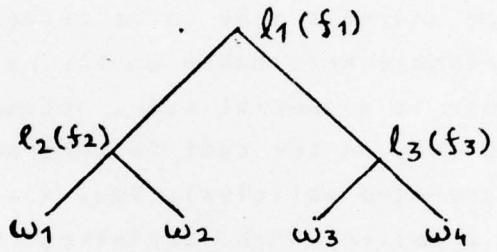


Fig. 3.2

If an OSMLR rule is used at each node, the total tree performance, and correct recognition rates at the nodes, are,

$$Pc(T)=P(w1).Pc(w1/l1).Pc(w1/l2)+P(w2).Pc(w2/l1).Pc(w2/l2)$$

$$+P(w3).Pc(w3/l1).Pc(w3/l3)+P(w4).Pc(w4/l1).Pc(w4/l3)$$

$$Pc(l1)=P(w1).Pc(w1/l1)+P(w2).Pc(w2/l1)$$
$$+P(w3).Pc(w3/l1)+P(w4).Pc(w4/l1)$$

$$Pc(l2)=P(w1).Pc(w1/l2)+P(w2).Pc(w2/l2) \;/\; [P(w1)+P(w2)]$$

$$Pc(l3)=P(w3).Pc(w3/l3)+P(w4).Pc(w4/l3) \;/\; [P(w3)+P(w4)]$$

Let x2=s be some state of feature f2, such that,

$$P(w1).p(x2=s/w1) > P(w2).p(x2=s/w2)$$

Then the OSMLR would classify a sample with x2=s as belonging to class w1. Now, suppose we choose the OPPOSITE of the maximum-likelihood rule and assign such a sample to w2. Then the change in Pc(l2), the recognition rate at node 2, is,

$$d(Pc(l2)) = P(w2)p(x2=s/w2) - P(w1)p(x2=s/w1)/[P(w1)+P(w2)]$$
$$< 0$$

Hence the node performance decreases, as expected. The change in the total tree's performance is,

$$d(Pc(T)) = P(w2)Pc(w2/l1)p(x2=s/w2) - P(w1)Pc(w1/l1)p(x2=s/w1)$$

The tree performance would IMPROVE if,

$$\frac{p(x2=s/w2)}{p(x2=s/w1)} > \frac{P(w1).Pc(w1/l1)}{P(w2).Pc(w2/l1)}$$

Hence, using a one-step maximum likelihood rule (OSMLR) at each node does not necessarily minimize the error rate of the total tree. The numerical example given below bears out this point.

Example:

Assume the decision tree of Fig.3.2 where f1 has four states, {a,b,c,d}, f2 can take on any of two states, {x,y}, and f3, {u,v}. The class conditional state probabilities of the features and aprior class probabilities are tabulated below.

		f1				f2		f3	
	P(wi)	a	b	c	d	x	y	u	v
w1	.3	.2	.2	.3	.3	.4	.6	.1	.9
w2	.2	.1	.6	.2	.1	.6	.4	.2	.8
w3	.4	.3	.3	.2	.2	.3	.7	.3	.7
w4	.1	.9	.1	.0	.0	.4	.6	.5	.5

Using an OSMLR at each node results in the following rules,

f1=a (go right)
=b (left)
=c (left)
=d (left)

f2=x (go right)
=y (left)

f3=u (go left)
=v (left)

The correct recognition rates of the tree and nodes, are,

Pc(T)=0.372 Pc(l1)=.63 Pc(l2)=.6 Pc(l3)=.8

If we violate the maximum likelihood rule at node 1 and go right for f1=b, the new performance figures are,

Pc(T)=.384 Pc(l1)=.58 Pc(l2)=.6 Pc(l3)=.8

Thus, the performance at node 1 has DECREASED, but the

total tree performance INCREASED when the OSMLR was violated.

3.4.1. A Bound On The Tree Performance

It was shown in the previous discussion that using a maximum likelihood rule at each node of a tree does not necessarily optimize the performance of the total tree, even when the node decisions are assumed to be statistically indpendent. It is clear ,however, that the choice of the node features places an upper bound on the optimal tree performance. In this section, the problem of computing this bound is solved as an optimization problem under linear constraints. The problem, so posed, is amenable to a host of solution techniques[32]. A method of using dynamic programming to obtain the upper bound is described here, using the example from the previous section for illustration.

Under the assumption of statistically independent node decisions, the tree performance is given by,

$$Pc(T) = \sum_{\omega_i} P(\omega_i) \cdot \prod_{l_k \in S(\omega_i)} Pc(\omega_i / l_k) \qquad \ldots\ldots\ldots\ldots (3.16)$$

The performance of the rule used at lk, averaged across all classes under lk is given by,

$$Pc(l_k) = \sum_{\omega_i \in W(l_k)} P(\omega_i) . Pc(\omega_i / l_k) \;\Big/\; \sum_{\omega_i \in W(l_k)} P(\omega_i) \qquad \ldots\ldots\ldots (3.17)$$

For a given choice of features used at lk, Pc(lk) is maximized if a maximum likelihood rule is used. Let bk denote the maximum value of the numerator of Eqn. (3.17), i.e.

$$b_k = \text{Max} \left(Pc(l_k) . \sum_{\omega_i \in W(l_k)} P(\omega_i) \right)$$

for ease of notation, let,

$P_i = P(w_i)$, $y_{ik} = P_c(w_i/l_k)$.

Then, the upper bound on $P_c(T)$ is obtained as the solution of the constrained optimization problem,

$$\max_{y_{ik}} \sum_{i=1}^{M} P_i \cdot \prod_{K} y_{ik} \qquad \text{.....(3.18)}$$

$$\sum_{\omega_i \in W(l_K)} P_i \cdot y_{ik} \leq b_k \quad , \; k=1,2,\ldots\ldots\ldots N .$$

$$0. \leq y_{ik} \leq 1. \quad , \qquad i=1,2,\ldots M, \; k=1,2,\ldots N .$$

where M ,N are the number of classes (terminal nodes) and decision nodes in the tree, respectively.

To solve (3.18), define the partial sum, Gn, n=1,2,...M, as

$$G_n = \sum_{i=1}^{n} P_i \cdot \prod_{K} y_{ik} \quad , \; 1 \leq n \leq M .$$

The problem can be solved by regarding the bk as the total resources available of each kind, and Gn as the return from the first n activities. Optimizing Gn, and then recursively defining Gn in terms of Gn-1, leads to the dynamic programming formulation.

Example:

Consider the tree below, consisting of three nodes, l1, l2, l3.

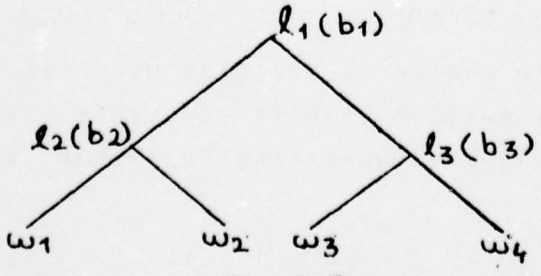


Fig.3.3

The problem is to maximize,

$$P1.y11.y12 + P2.y21.y22 + P3.y31.y33 + p4.y41.y43$$

subject to,

$$P1.y11 + P2.y21 + P3.y31 + P4.y41 \leq b1$$
$$P1.y12 + P2.y22 \leq b2$$
$$P3.y33 + P4.y43 \leq b3$$
$$0. \leq yik \leq 1.$$

Define the following sequence of optimization problems:

$G1(z1,z2) = \text{Max } P1.y11.y12$ for $0. \leq z1 \leq b1$, $0. \leq z2 \leq b2$.

where, $0. \leq y11 \leq \text{Min}\{1., z1/P1\}$

$0. \leq y12 \leq \text{Min}\{1., z2/P1\}$

$G2(z1,b2) = \text{Max}\{P2.y21.y22 + f1(z1-P2.y21, b2-P2.y22)\}$

for, $0. \leq z1 \leq b1$,

where, $0. \leq y21 \leq \text{Min}\{1., z1/P2\}$

$0. \leq y22 \leq \text{Min}\{1., z2/P2\}$.

$G3(z1,b2,y3) = \text{Max}\{P3.y31.y33 + f2(b1-P3.y31, b2)\}$

for $0. \leq z1 \leq b1$, $0. \leq z3 \leq b3$,

where $0. \leq y31 \leq \text{Min}\{1., z1/P3\}$

$0. \leq y33 \leq \text{Min}\{1., z3/P3\}$

$$\text{Max } P_c(T) = G4(b1,b2,b3) = \text{Max}\{P4.y41.y43 + G3(b1-P4.y41, b2, b3-P4.y43)\}$$

where $0. \leq y41 \leq \text{Min}\{1., b1/P4\}$

$0. \leq y43 \leq \text{Min}\{1., b3/P4\}$.

AD-A042 161 MARYLAND UNIV COLLEGE PARK DEPT OF COMPUTER SCIENCE F/G 9/4
OPTIMAL AND HEURISTIC SYNTHESIS OF HIERARCHICAL CLASSIFIERS.(U)
AUG 76 A V KULKARNI AF-AFOSR-2901-76
UNCLASSIFIED TR-469 AFOSR-TR-77-0825 NL

2 OF 2
AD A042161

END
DATE FILMED
8-77
DDC

The problem was solved by discretizing the ranges of the variables, zj and yik into 10 equal intervals. The results are tabulated for different apriori probabilities, and resource levels, bk. The relationship between the optimal tree performance and a given node's performance when the other node performances are kept fixed, is depicted by the plots in Graph 1.

Variation Of Pc(T) With Node Performance

Pc(T) : Tree performance
b1 : Node 1 performance
b2 : Node 2 performance
b3 : Node 3 performance

b2 \ b3	0.2	0.4	0.6	0.8	1.0
0.2	.160	.234	.278	.314	.354
0.4	.234	.280	.317	.355	.390
0.6	.278	.317	.354	.392	.427
0.8	.314	.355	.392	.430	.465
1.0	.354	.390	.427	.465	.500
0.2	.216	.310	.365	.437	.502
0.4	.310	.396	.446	.481	.540
0.6	.365	.446	.497	.530	.576
0.8	.437	.481	.530	.563	.611
1.0	.502	.540	.576	.611	.650
0.2	.255	.355	.422	.517	.592
0.4	.355	.455	.521	.602	.672
0.6	.422	.521	.578	.653	.723
0.8	.517	.602	.653	.695	.761
1.0	.592	.672	.723	.761	.800
0.2	.290	.390	.462	.562	.637
0.4	.390	.490	.562	.662	.737
0.6	.462	.562	.630	.729	.804
0.8	.562	.662	.729	.810	.880
1.0	.637	.737	.804	.880	.950
0.2	.300	.400	.475	.575	.650
0.4	.400	.500	.575	.675	.750
0.6	.475	.575	.650	.750	.825
0.8	.575	.675	.750	.850	.925
1.0	.650	.750	.825	.925	1.00

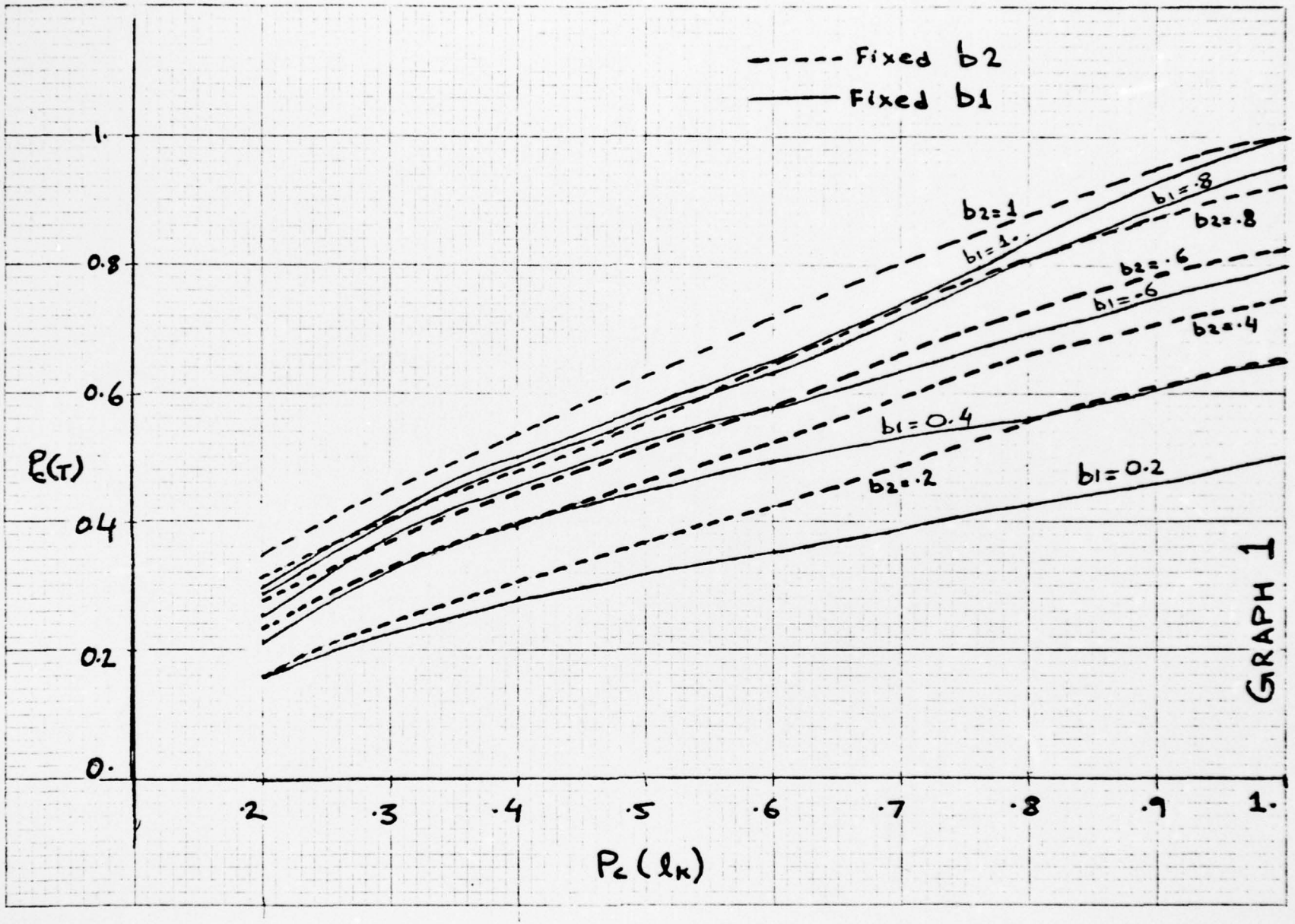
3.5. A Property Of The Optimal Feature Assignment

In the previous section, it was shown that optimizing separately, the average correct recognition rate at each node of a decision tree, does not necessarily optimize the total tree performance. A similar type of 'negative' property can be shown for the optimal feature set.

Assume that we use only a maximum likelihood rule at each tree node. One could evaluate each feature's 'goodness' at each node as the value of Pc(lk) when that feature is used at node lk. Then the assignment could be done by assigning that feature to each node, which had the best value of the goodness measure at that node. Property 2 which we show by example, highlights the fact that such an assignment is not necessarily optimal.

Property 2:

The tree resulting from using at each node, that feature which performs the best at that node using a maximum likelihood rule, is not necesarily the optimal assignment, even if the features are statistically independent.



Example:

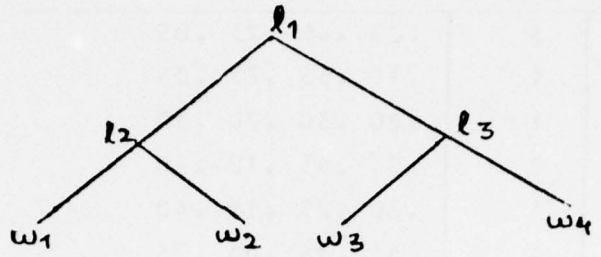


Fig. 3.4

Consider the 2 level binary decision tree shown in Fig.3.4. The features to be used at the nodes have to be chosen from a set of 6 features, f1-f6, each taking on one of 4 states. The probability of occurrence of each state, for each class, is given in the table below:

FEATURE	CLASS	STATE PROBABILITIES
1	1	.20 .20 .30 .30
1	2	.30 .22 .38 .10
1	3	.20 .50 .10 .20
1	4	.60 .10 .10 .20
2	1	.40 .10 .20 .30
2	2	.40 .50 .10 .00
2	3	.20 .50 .20 .10
2	4	.10 .40 .30 .20
3	1	.45 .10 .05 .40
3	2	.40 .10 .35 .15
3	3	.50 .20 .20 .10
3	4	.60 .10 .20 .10
4	1	.30 .10 .50 .10
4	2	.50 .20 .10 .20
4	3	.15 .35 .45 .05
4	4	.15 .15 .15 .55
5	1	.20 .40 .20 .20
5	2	.20 .30 .40 .10

5	3	.25 .45 .25 .05
5	4	.15 .55 .25 .05
6	1	.20 .30 .20 .30
6	2	.35 .45 .10 .10
6	3	.20 .25 .15 .40
6	4	.55 .15 .05 .25

The apriori class probabilities are, .15,.1,.50 and .25, respectively.If a maximum likelihood rule is used at each node to partition the set of classes into the two groups distinguished at that node, the correct recognition rates, using each of the 6 features, is tabulated below:

NODE	F1	F2	F3	F4	F5	F6
1	.76	.75	.75	.75	.75	.75
2	.60	.74	.71	.66	.64	.62
3	.73	.67	.67	.82	.67	.72

From the above table, it would appear that the assignment of feature 1 to node 1, feature 2 to node 2, and feature 4 to node 3, would result in the least probability of error for the total tree, assuming a maximum likelihood rule is used at every node.However, as the figures below prove, there are other features, which together do better than the set {f1,f2,f4} used at nodes {1,2,3}, respectively.

Feature used at Node1	Node2	Node3	Recognition Rate
1	2	4	.611
2	4	4	.613
2	5	4	.613
3	4	4	.613
2	6	4	.613

Thus, even using suboptimal features, 2 and 5 at nodes 1 and 2, we get a better error rate than using the individually best features at these nodes, viz., 1 and 2.

In this chapter, it has been shown that even under the restriction that the node decisicns are statistically independent, th optimal tree design problem is complex. The next two chapters discuss methods of decomposing the design problem into phases, and the use of optimization methods in solving each phase of the task.

4. A Phased Approach To Optimal Tree Design

This chapter proposes a decomposition of the hierarchical classifier design problem into several phases, and investigates the use of dynamic programming procedures for obtaining the optimal solution for each phase.

The decomposition of the design allows the designer to input into the design procedure any a priori knowledge regarding the problem that is available. This knowledge might be related to the tree structure or the features deemed to be 'good' for distinguishing certain classes. The a priori information regarding the tree can be categorized into three levels:

(a) No assumptions regarding the form of the tree.

(b) The tree 'skeleton' is assumed given. By 'skeleton', we mean, the form of the tree together with terminal node labels.However, the features to be used at each node are not specified.

(c) Both the tree skeleton and the features to be used at the nodes, are specified.The decision rules at each node have to be designed.

The type of assumptions, (a), (b), or (c), made in any particular application depend upon the designer's knowledge about the data: its modality, separability, and the goodness of particular features in distinguishing some subsets of the data. Thus, in a general situation, when one's problem knowledge is minimal, only assumption (a) may be justified. If through data analysis methods, such as hierarchical clustering or apriori knowledge, a suitable tree skeleton is considered adequate, assumption (b) may be used. If, in addition, the designer knows that certain features are good for discriminating between some two or more subsets of the

classes, one also knows the features to be used at some or all of the tree nodes. The design problem is then that of case (c), where the decision rule at each node must be specified for overall optimal tree performance.

The following analysis shows that finding the optimal tree by exhaustive enumeration is an impractical approach for most problems.

4.0 Computational Complexity

The tree design problem is a 'three-dimensional' search involving

(i) specifying a tree skeleton

(ii) assigning features to nodes, and,

(iii) specifying decision rules at each node.

Consider the number of trees to be evaluated if an exhaustive search is done across all possibilities in (i) - (iii). If Nt, Nf, Nd, are respectively, the possible trees, assignments of features to each tree, and decision rules for each assignment, then the total possibilities are,

$$N = Nt.Nf.Nd$$

As an example, consider a C class problem, using F features, each feature taking on one of S states, and with the restriction that only balanced binary trees, with a single path to each class, and using any feature only at one node, are being considered. Then,

$$Nt = \prod_{n=1}^{\log_2 C} \left[\frac{1}{2} \binom{c/n}{c/2n} \right]^{2^{n-1}}$$

$$Nd = 2^{CS-S}$$

$$Nf = \binom{F}{C-1} \cdot (C-1)!$$

The possibilities become enormous even for 'small' values of F, C, and S. The computational complexity of the problem is considerable reduced by using the optimality principle of dynamic programming.

4.1. Tree Design Using Dynamic Programming

In this section, we examine a set of methods that have proved useful in operations research and popularly known as dynamic programming[2].

The criterion of optimality is assumed to be a weighted sum of the measurement cost and the risk of making an incorrect classification. Depending upon the information available to the designer regarding the tree, the design problems may be posed in the following way, in increasing order of complexity.

(1) Given the tree structure (i.e. the class hierarchy and the features to be used at the nodes), design an optimal policy.

(2) Given only the tree skeleton, (i.e. the class hierarchy), determine the optimal feature measurement and decision policy.

(3) Given no information regarding the form of the tree, design an optimal tree structure.

The algorithms proposed in subsequent sections for solving (1) and (2) give the optimal solution for any assumed feature distribution. The bottom-up procedure for problem (3), however, leads to the optimal tree structure only under an 'additive cost' assumption described later.

4.1.1. Optimal Decision Policy Given The Tree

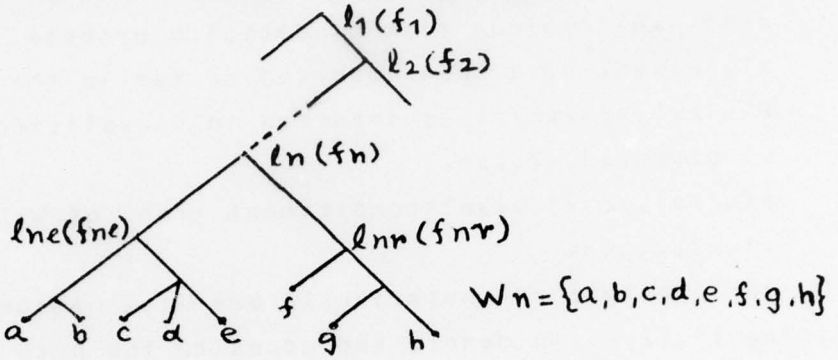


Fig. 4.1

Consider the case when the tree structure of the classifier is given and one seeks the optimal decision policy at each of the nodes. One would also like to have the facility of terminating the measurement process at any time and 'accept' one of the class labels which has NOT been rejected till that point. In a decision tree, some classes are rejected at every node. Thus, the design procedure must compute at each node the optimal action to be taken, and this can be one of the following :

- classify the sample into some class not rejected so far or,
- continue the measurement process and decide the next node to be traversed under the current node.

The following quantities will be defined to illustrate the dynamic programming (abbreviated DP) procedure, using the example of Fig. 4.1 above.

ℓ_n = node label at which feature f_n is used,
f_{nl}, f_{nr} are the features used at the nodes to the

left/right below ln(assuming a binary decision node),

c(fn)=cost of measuring feature fn

r(x1..xn)=minimum risk of decision process, given that x1,x2..xn, have been observed so far in the tree.

R(wi/x1,x2..xn)=loss incurred in classifying sample into wi given x1,x2..xn.

p(wi/x1,x2,x3...xn)=conditional prob. of wi given x1,x2,...xn.

For the sake of notational brevity, assume that the string l1,l2,...ln denote the nodes on the path to ln from the root.If lnl,lnr denote the sons of ln and the corresponding feature observations are xnl,xnr, then, the minimum risk r(x1,x2,... xn) may be computed recursively as,

$$r(x1,x2...xn)=\text{Min}\begin{cases}\text{(i)}\ \ \underset{\omega_k\in W_n}{\text{Min}}\ R(w_k/x1,x2..xn) & \text{.....(4.1)}\\ \text{(ii)}\ \ c(fnl)+E\{r(x1,x2...xn,xnl)\}\\ \text{(iii)}\ \ c(fnr)+E\{r(x1,x2..,xnr)\}\end{cases}$$

where, the E{} terms are expectations of the risk, r() over all possible values of xnl, or xnr, respectively.

If ln is a terminal node at which the decision is to classify the sample in class wk, then,

$$r(x1,x2..xn)=R(w_k/x1,x2..xn)$$

Working in a bottom-up fashion from the terminals, we can determine the optimum policy and risk at each node.In the recursive equation (4.1), if the quantity (i) is minimum, then it implies that it is best to classify the sample and discontinue further measurement, while if (ii) or (iii) are the smallest, it implies that the best action is to traverse the left or right node respectively. Correspondingly, one set or the other set of classes is

rejected from further consideration.

Computationally, the decision tree formulation is less burdensome than the sequential process described in [5], where at each step, R(wk/x1..xn) must be computed for all classes, wk.In the decision tree, the number of classes to be considered, increases from one(at a terminal node) to M(i.e. the total # of classes), at the root.

Note, that if all measurement costs,c(fi), are zero, the quantity in (i) can never be the smallest, since more measurements cannot increase the risk(assuming perfect knowledge of p(x1,..xN/wi). Hence, if measurement costs are zero, an optimum policy will only make the classification at a terminal node.

4.1.2. Optimal Feature Ordering and Decision Policy

In this section, we consider the case where the tree skeleton is given, but the features to be measured at the nodes are not specified.Thus, the optimal policy must specify not only the action to be taken , but also the best feature to be measured next, in such a manner that the average cost (measurement plus misclassification risk) is minimized. Since the tree form is given, the classes at each node that have not been rejected, are known. Hence, for a given set of values,xt1,xt2,...xtn, of the features in the set Ftn$\subset$ FN, the total feature set, we must decide:

(i) whether the sample should be classified into a class that has not been rejected so far, or

(ii)reject one set(left subtree's terminals) or the other from consideration, and,
decide the feature to be measured at the next node.

Let,

Ftn=set of n features.

rn(xt1,xt2,..xtn/Ftn)=minimum cost of making a decision at node ln, given the particular observed values of the particular feature set, Ftn.

R(wi/xt1,xt2..xtn,Ftn)=loss incurred by classifying sample into class wi, given observations on features in Ftn.

Then, rn() can be computed in a backward fashion using,

$$rn(xt1,..xtn/Ftn)=\text{Min}\begin{cases}\text{Min } R(wj/xt1..xtn,Ftn) & ...(4.2)\\ \quad wj\in Wn & \\ \text{Min } c(fk)+E(rnl(xt1..xtn,xk)) & \\ fk\notin Ftn & \\ \text{Min } c(fk)+E(rnr(xt1..xtn,xk)) & \\ fk\notin Ftn & \end{cases}$$

where rnl,rnr are the minimum costs at the left and right
sons of node ln.
For terminal node, ln,

$$rn(xt1,xt2,..xtn/Ftn)=R(wk/xt1..xtn,Ftn)$$

The computational complexity is considerably increased by the necessity to consider all subsets of n features at any node n levels below the root.

4.1.3. Optimal Tree Structure

In many situations, one has no a priori information regarding a 'good' tree structure. The design could then be split into two phases. In the first phase, the optimal tree structure is designed, assuming for example, that a maximum likelihood rule is used at every node. In the second phase, the procedure of Sec.4.1.1 can be used to determine the optimal decision policy at every node. This section describes a 'bottom-up' design method for obtaining the optimal tree structure. Sufficient conditions for the optimality of the resulting structure are also discussed.

Let,
W=total set of classes(=M)
$W_k \subset W$ denotes a k-class subset of the M classes,
T(W)= tree used to separate the classes in W.
The labels (feature names) used at the nodes, complete the tree description.

T*(W)=optimal tree on W, i.e. one that minimizes cost.

The 'cost' of a decision tree, r(T(W)), is given by,

$$r(T(W)) = \sum_{w_i \in W} P(w_i)\left[\sum_{l_k \in S(w_i)} c(f_k) + \left(1 - \prod_{l_k \in S(w_i)} (1 - Pe(w_i/l_k))\right)\right] \quad \ldots\ldots (4.3)$$

where,
Pe(wi/lk)=probability of error at node lk if sample is from class wi.

The following assumptions have been made in eqn.(4.3).

(i) the prob. of correct recogniton of a random sample from wi, is a product of the prob. of correct recognition at the nodes on the path to wi.

(ii) the measurement cost of misclassified samples is negligible compared to that of correctly classified samples.

A sufficient condition for (i) to be valid is that the features on any path in the tree fom the root to a terminal node are statistically independent. Condition (ii) is valid if the error rate of the tree is low.

The recursive definition of the optimal tree, $T^*(W_k)$, to classify the k-class set W_k, is written as,

$$r(T^*(W_k)) = \min_{\forall T(W_k)} [r(T(W_k))]$$

$$= \min_{\forall f} \left[\min_{\forall W_1, W_2} \left\{ P(w_i)(c(f_k) + Pe(w_i/l_k)) + r(T^*(W_1)) + r(T^*(W_2)) \right\} \right] \quad \text{.....(4.4)}$$

where $W_1 \cup W_2 = W_k$

In the above formula, the cost of the tree, $T(W_k)$, has been written as the sum of the costs incurred at the top node (the summation term), plus the costs of the optimal trees for sets W1 and W2, viz., $T^*(W_1)$ and $T^*(W_2)$. In the next section, we provide a justification for this assumption of additive costs. Under these conditions, it is shown that the recursive definition (4.4) results in the optimal solution tree, $T^*(W)$. Note that in (4.4), the error rate for wi at node lk, $Pe(w_i/l_k)$, depends not only on the feature

used at lk, but also on the dichotomy of classes that is performed(i.e. W1 versus W2).

4.1.3.1. The Additive Cost Assumption

Assume that the features used at the nodes on a path in the tree are statistically independent, and that the errors, Pe(wi/lk), are small.Then, terms such as Pe(wi/lk).Pe(wi/lj) can be ignored in comparison to the first order terms. If this done, the error rate of the tree, Pe(T), is given by,

$$Pe(T) = \left(1 - \sum_{wi} P(wi) \cdot \prod_{l_k \in S(\omega_i)} (1 - Pe(wi/lk))\right)$$

$$= \sum_{wi} P(wi)\left(1 - \left(1 - \sum_{lk} Pe(wi/lk) + \text{higher order terms}\right)\right)$$

where $lk \in S(wi)$.

$$= \sum_{wi} P(wi) \sum_{lk \in S(\omega_i)} Pe(wi/lk)$$ ignoring higher order terms.

Hence, the total error probability is roughly the sum of the error rates for each class at all the nodes on the path to that class. If we also assume that the measurement cost on misclassified samples is negligible compared to the total cost, we get,

$$r(T(W)) = \sum_{wi \in W} P(wi) \sum_{lk \in S(wi)} \{c(fk) + Pe(wi/lk)\} \quad \ldots\ldots\ldots\ldots (4.5)$$

It follows immediately from (4.5) that the cost of a decision tree is the sum of the cost incurred at the root node plus the costs incurred in each subtree below the root.Hence, we are justified in writing (4.4) as the sum of the cost terms.Therefore, because the costs are additive,

minimizing each term would minimize the total cost. Therefore, the recursive formula,(4.4), leads to the globally optimal tree for the set of classes, W.

The following theorem generalizes the conclusion reached in the above discussion and gives sufficient conditions on the criterion function, Pc(T) of tree goodness, under which the dynamic programming algorithm for tree design described in the preceding section will be optimal.

Theorem

Let Pc(T) be the goodness measure of a binary tree used to separate a set of classes, W. Let l0 be the root of T, f0 the feature used at l0, and T0(l0) and T1(l0) the left and right subtrees below node l0. Assume that Pc(T) can be written as the sum of three terms as follows:

$$Pc(T) = Pc(T0(l0)) + Pc(T1(l_0)) + G(W0,W1,f0)$$

where the first two terms are the goodness measures of the two subtrees T0(l0) and T1(l0), and the term G depends ONLY on the feature f0 used at l0, and the sets of classes W0, and W1 distinguished at l0, but NOT on the tree structures T0 and T1. Then the algorithm of Sec.4.2.3 finds the optimal binary tree, T*, to separate the classes, W, under the assumption that each class occurs at only one terminal node of T*.

Proof:

Let W′ be any subset of classes of the total set, W, F be the total set of features. Let T*(W′) be the optimal binary tree(i.e. the one that maximizes Pc(T(W′)), under the assumption that each node uses a single feature, and each class occurs at only one terminal node of the tree. Then

under the assumption that Pc(T) can be written in the above form, it follows that,

$$Pc(T^*(W')) = \max_{\forall T(W')} \{ Pc(T(W')) \}$$

$$= \max_{\forall f} \max_{\forall T(W_0),T(W_1)} [Pc(T(W0)) + Pc(T(W1)) + G(W0,W1,f)]$$

where, $W0 \cup W1 = W'$, $f \in F$, and T(W0) and T(W1) are the subtrees of any tree T for classes W', and have terminal classes as W0 and W1.

Since the G term does not depend on the trees T(W0), T(W1), but only on the sets, W0 W1 separated at the top node of T(W'), and since the first two terms can be maximized separately, we get,

$$Pc(T^*(W')) = \max_{\forall f} \max_{\forall W_0,W_1} [Pc(T^*(W0)) + Pc(T^*(W1)) + G(W0,W1,f)]$$

where W0 W1 = W' and $f \in F$.

The above equation is precisely the recursive form used by the algorithm of Sec. 4.1.3., as it builds the optimal tree T*(Wk) for k class subsets Wk of W. Since T*(Wk) is optimal, it follows that T*(W), the final tree obtained, must also be optimal.

It is seen that the distance measures discussed in Sec.3.2.2 satisfy the conditions of this theorem and hence the same algorithm could be used to obtain an optimal tree using the criterion given by equation (3.4) in that section.

This chapter has shown the feasibility of using the dynamic programming formulation for solving each of the three phases of tree design. The solutions obtained are

optimal under fairly general assumptions. Thus, the optimal decision policy and optimal feature ordering are obtained without making any particular assumptions regarding the fearture distributions, such as statistical independence. The optimality of the tree structure obatined via the 'bottom-up' procedure has been established under an additive cost assumption.

In spite of these general conditions for optimality, dynamic programming methods suffer from a high computational complexity when applied to 'large' problems. In the context of tree design, a large problem is one where the number of classes to be distinguished is large, or the features from which the selection of 'good' features is desired, are numerous. The next chapter investigates methods of reducing this design complexity.

5. Methods Of Reducing The Computational Complexity

This chapter proposes various methods of reducing the computational burden incurred in obtaining the the optimal decision policy for each node of a given tree structure and the optimal feature assignment for each node of a given tree skeleton.

The reduction in computations incurred in determining the best decision policy is achieved in the following ways:

(1) It is assumed that the node decisions are statistically independent
(2) The decision at a node is computed only as a function of the feature measurement at that node.
(3) For non-metric features, the decision rules can be 'clustered' and each cluster represented by a prototype. The DP procedure can then be used to search across all prototypes rather than all rules.
(4) Many decision rules can be discarded from consideration if they are 'dominated' by others. Thus only the set of 'efficient' rules need to be searched. A branch-and-bound algorithm for finding the set of efficient rules is described.

Reduction in the complexity involved in getting the best assignment of features to the nodes of a given tree skeleton, is achieved as follows:

(1) It is assumed that the decisions at the nodes are statistically independent;
(2) the choice of the feature to be measured at each node is fixed once the tree is designed, i.e. this

choice is not a function of the feature values observed for a test sample;

(3) at each node, the features can be ranked and certain features discarded, thus reducing the search dimensionality.

5.1. Decision Policy Design Given The Tree

A dynamic programming method for obtaining the optimal decision policy was described in Chapter 4. In this bottom-up approach, the risk was calculated at each node as a function of the past measurements and the optimal risk from this point onwards to the terminals below that node. Consider a path in the tree through nodes, $(l_1, l_2 \ldots l_k)$, to classes w_1, w_2, as shown in Fig. 5.1 below.

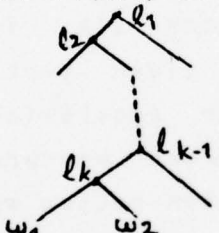


Fig. 5.1

In the DP approach, we compute $r(x_1, x_2, \ldots x_k)$ as the minimum misclassification risk at l_k, for all sequences, $(x_1, x_2, \ldots x_k)$. Then we back up to l_{k-1} and choose the decision (go left or right) for each possible subsequence, $(x_1, x_2, \ldots x_{k-1})$, and so on up the tree. To compute the decision the entire past history of measurements and the optimal decision sequence among all subsequent decision paths, are considered. In this way, the globally optimal policy is obtained.

At the other extreme, one could use only the last measurement to make the decision. This corresponds to using a 'one-step' policy, as described in Chapter 3. It was

shown there, that using a maximum likelihood policy (OSMLR) to optimize each node's correct recognition rate, does not, in general, optimize the total tree's performance.This is to be expected, since in the OSMLR methcd, no use is being made of the history of previous measurements, or the risk of subsequent decision paths.

A question that arises is: between these two extremes, viz. DP which uses all the 'past', and projects into all possible 'future' decisions, and the OSMLR, which uses only the 'present', can one steer a middle course, and use some 'compact' summary of past measurements, rather than the measurments themselves. Such a strategy may do at least as well as OSMLR, while simultaneously reducing the computation and storage requirements of the DP algorithm.

One such 'compact' measure of previous observations, given that one is at node ,lk, is the a posteriori likelihood of each class, given that we have traversed nodes, l1,l2,...lk-1. The aposteriori likelihood is a function of the feature values measured at the previous nodes, as well as the decision policy used at those points, and it has a simple form when the features are statistically independent. The following theorem proves that such a 'condensed-history decision rule' (abbreviated CHDR), performs at least as well as an OSMLR policy, and is bounded above by the optimal solution obtained by DP methods.

Theorem:

Assume that all features along the path of Fig.5.1 are statistically independent Consider the following three ways of computing the decision policy at node, lk.

DP: If xk=s is observed, and,
if P(w1).p(x1,x2,..xk-1,s/w1)
> P(w2).p(x1,x2...xk-1,s/w2)
then classify X in w1, else in w2.

OSMLR: If xk=s is observed, and
if P(w1).p(xk=s/w1) > P(w2).p(xk=s/w2)
then classify X in w1, else in w2.

CHDR: Let Pr(l1,l2..lk-1/wi) be the probability that a sample from wi will arrive at lk, and
if xk=s is observed and,
if P(w1).p(xk=s/w1).Pr(l1..lk-1/w1)
> P(w2).p(xk=s/w2).Pr(l1,..lk-1/w2)
then classify X in w1, else in w2.

If Pc(DP), Pc(OSMLR), Pc(CHDR), are the average performance (probability of correct recognition), of each rule when used at lk,given the same decison rules at all previous nodes, then,

$$Pc(OSMLR) \leqslant Pc(CHDR) \leqslant Pc(DP)$$

Proof:

Let d1,d2,..dk-1, represent the decision vectors at nodes, l1,..lk-1, and X(d1), X(d2),..X(dk-1), the sets of samples that would be passed down this path by each rule. Then, the set arriving at lk, is,

$$X(d1) \cap X(d2) \cap \ldots \cap X(dk-1) \equiv X(l1,l2..lk-1)$$

The probability that a random sample from class wi will arrive at node lk, is given by,

$$Pr(l1,l2..lk-1/wi) = \sum_{X \in X(\ell_1,\ell_2,..\ell_{k-1})} p(X/wi) \quad .$$

The correct recognition rates of the three rules can be written as,

$$Pc(DP) = \sum_{X} \sum_{x_k} Max\{ P(w1).p(X/w1).p(xk/w1), \; P(w2).p(X/w2).p(xk/w2) \}$$

where $X \in X(d1,d2,\ldots..dk-1)$

$$Pc(OSMLR) = \sum_{x_k} p(X(d1,d2,..dk-1)/w*) \; .Max\{ P(w1)p(xk/w1), \; P(w2)p(xk/w2) \}$$

where $w* = w1$, if $P(w1)p(xk/w1) > P(w2)p(xk/w2)$

$= w2$ otherwise.

$$Pc(CHDR) = \sum_{x_k} Max\{ P(w1).p(xk/w1).Pr(X(l1,l2,..lk-1)/w1), \; P(w2).p(xk/w2).Pr(X(l1,l2..lk-1)/w2) \}$$

From the algebraic inequality,

$$\sum_{j} \sum_{i} Max\{ai.bj, ci.dj\} \geq \sum_{j} Max\{bj \sum_{i} ai, dj \sum_{i} ci\}$$

and by equating,

ai, ci, to p(X/w1), p(X/w2), respectively,

$\sum_{i} ai$, $\sum_{i} ci$ to Pr(X(l1,l2..lk-1)/w1) and Pr(X(l1..lk-1)/w2), and bj, dj to P(w1).p(xk/w1), P(w2)/p(xk/w2) it follows that,

$$Pc(DP) \geq Pc(CHDR)$$

Similarly, by using the fact that,

$$\sum_{j} Max(a1.bj, a2.cj) \geq \sum_{j} a*.Max(bj, cj)$$

where $a*=a1$, if $bj \geq cj$

$= a2$ otherwise,

it follows that,

$$Pc(CHDR) \geq Pc(OSMLR)$$

This theorem suggests the use of the measure, Pr(X(l1..)/wi), in finding the optimal policy at lk, viz., dk. If each feature has m states, a decision vector di has m components, dij, j=1,2,..m, where, dij=n implies that if fi is in state j, the n th. son of the current node is to be traversed next. For statistically independent features, and particular decisions, d1,d2..dk, one can write,

$$Pr(X(l1,..lk-1)/wk) = \prod_{di} Pr(X(di)/wk)$$

$$= \prod_{di} \left[\sum_{j=1}^{m} p(xi=j/wk) \right]$$

$$= \prod_{di} gk(di)$$

where gk(di) is the summation within the square brackets.

This sum is taken over all values of j such that dij=n, and the class wk is under the n th. son of node li, e.g. in a binary tree, $dij \in \{1,2\}$, and in Fig.5.1, and for node lk, and class w1, the summation would be over all j such that dkj=1.

The history of previous measurements, is thus retained in the g functions, since their product represents the probability that a sample from class wi will arrive at node

lk. The DP method can be used to evaluate the optimal one-step policy at each node, but unlike in the Chapter 4 formulation, the decision di, at each node is derived as a function of the decision vectors used at the ancestor nodes of li, rather than the actual measurements made at these nodes, xl1,xl2, etc.

5.1.1. Optimal One-Step Decision Policy

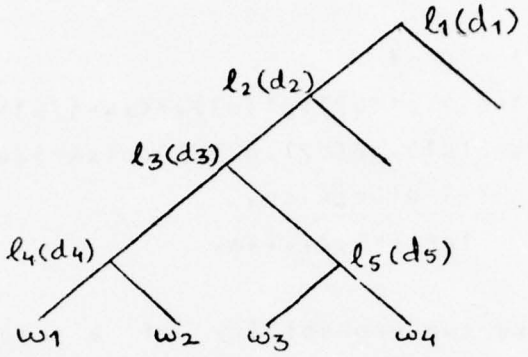


Fig. 5.2

The decision tree of Fig.5.2 shows the nodes, li, and the decision vectors, represented by di, used in classifying a subset of classes, viz. {w1,w2,w3,w4}.A dynamic programming procedure can be used to derive the optimal 'one-step' policy at each node. For statistically independent features, the correct recognition rates of the classes can be written in terms of the g functions introduced in the last section.

$$Pc(w1)=g1(d1).g1(d2).g1(d3).g1(d4)$$
$$Pc(w2)=g2(d1).g2(d2).g2(d3).g2(d4)$$
$$Pc(w3)=g3(d1).g3(d2).g3(d3).g3(d5)$$
$$Pc(w4)=g4(d1).g4(d2).g4(d3).g4(d5)$$

The total tree performance is,

$$Pc(T)=\sum_{\omega_i} P(wi).Pc(wi) \qquad \text{.....(5.1)}$$

Assume each feature has m states. One can compute the optimal decision rule d4* as a function of d1,d2,d3, as

$$d4j^*(d1,d2,d3) = 0 \text{ if}$$
$$P(w1).g1(d1).g1(d2).g1(d3).p(x4=j/w1)$$
$$> P(w2).g2(d1).g2(d2).g2(d3).p(x4=j/w2)$$
$$= 1 \text{ otherwise,}$$
$$\text{for } j=1,2,\ldots m.$$

This rule uses the probability of a class wi sample arriving at node 4, viz., gi(d1)gi(d2)gi(d3), and the observation x4=j, to make the decision. Once d4* and d5* have been calculated for all d1,d2,d3, one can back up to node 3, and compute d3* as a function of d1,d2, as that value of d4 which maximizes the quantity,

$$\begin{aligned} \max_{d3} \; & P(w1).g1(d1).g1(d2).g1(d3).g1(d4^*)+ \\ & P(w2).g2(d1).g2(d2).g2(d3).g2(d4^*)+ \\ & P(w3).g3(d1).g3(d2).g3(d3).g3(d5^*)+ \\ & P(w4).g4(d1).g4(d2).g4(d3).g4(d5^*)\ldots\ldots. \end{aligned} \quad (5.2)$$

In the above sum, note that the values of d4, d5 substituted are those obtained in the last step, viz. d4*(d1,d2,d3), and d5*(d1,d2,d3). Working up the tree, one can finally compute the optimal decision vector at l1, d1*. Once this is done, d2* is found from the table of d2*(d1) computed during the backward procedure, d3* from the table, d3*(d1,d2), etc. The resulting decision rules are optimal one-step rules, though the globally optimal rules can be obtained, in general, only by using all the measurements already observed, as described in Chapter 4.

While giving up optimality, the above procedure has resulted in a considerable saving in storage as far as the

FINAL decision tree is concerned. The final tree, has stored at each node, an m-word decision vector for the m possible states that can be observed at that point. For the globally optimal solution, one would have to store at a node, a decision vector for every possible sequence of measurements, that may lead to that node. For a node n levels deep in the tree, and assuming that approximately half (binary tree assumed) the decisions are to go left/right, the storage of the decision policy at that node would require the following number of words:

$$\left(\frac{m}{2}\right)^n$$

However, there is no saving in computing or storage cost in this method, as compared to the general DP method of Chapter 4, since both methods compute the optimal decision as a function of every possible sequence of previous observations/decision vectors during the backward procedure.

Two techniques for reducing this cost are described in the next section. They use the g functions to transform the search domain from that of the discrete non-metric space of the decision vectors, di, to the metric space defined by the real-valued functions, gk(di).

5.1.2. Clustering in Decision Space

In the previous section, it was shown that dynamic programming can be used to recursively specify the optimal decision vector at each node of a hierarchical decision tree. When the features used at each node are discrete and non-metric, taking on one of m states, the decion vector for a binary tree, is a m-dimensional binary vector.Its i th. element, di, specifies whether to traverse the left or right subtree below that node when the feature is in state i.Thus,

at every node, one must choose from among the set of $(2)^m$ possible vectors,d, denoted by D. For a node n levels deep, the DP procedure would have to evaluate $(2)^{mn}$ possibilities in order to compute the optimal policy at that node. This 'mushrooming' of the computations as a function of m and n, makes the DP method impractical for all but very simple problems.

This section presents a method of grouping the decision vectors, $d \in D$, into sets, D1, D2,.....Dk, and choosing compact representation of each set, Di, by a prototype vector, ti.If the decision space at the i th node is partitioned into mi sets, and each set represented by a single prototype, the DP procedure searches for the optimal rule only from amongst the prototypes. Thus, the computations needed to find the optimal policy at a node n levels deep, are reduced to,

$$\prod_{k=1}^{n} mk \qquad \text{which is less than } (2)^{mn} .$$

The reduction in computations is gained at the expense of a departure from optimality, since only the prototype vectors are searched by the dynamic programming procedure.The prototype vector represents more than vector. This is achieved by specifying some of its m bits to be 'don't care' bits rather than 0 or 1.The selected don't care bits are such that varying them keeps the recognition rate for all M classes within a suitable tolerance interval. Thus, two vectors, d and d' from the same cluster differ in their performance (for each of the M classes), only in the specified margin. The analog of this clustering (grouping) concept, for the case of continuous valued variables, is that of discretizing the variables for use in dynamic programming.[2]

The next section describes a similarity measure for decision vectors. This measure will be used subsequently for clustering the vectors.

5.1.2.1. Similarity Measure For Decision Vectors

Two decision vectors which are candidates for use at a node l, are similar if exchanging one with the other does not change the tree performance by a significant amount. This simple definition of similarity can be used to cluster decision vectors which can be used at l. Assume for simplicity that a binary decision is made at l. Also let,

m: number of states that can be assumed by feature used at l,

di: a binary decision vector of m components, dij, j=1,2,...m.

D: the total set of decision vectors di, i=1,2,...(2) that can be used at node l.

Pc(wk/di): correct recognition rate for class wk if rule di is used at the node.

pk(j): probability that in class wk the feature is in state j.

Hence,

$$Pc(wk/di) = \sum_{\{j: d_{ij}=0\}} pk(j) \quad \text{if } wk \in W0(l)$$

$$Pc(wk/di) = \sum_{\{j: d_{ij}=1\}} pk(j) \quad \text{if } wk \in W1(l)$$

ti: a prototype vector whose m elements, tij, j=1,2,..m, can have values '0' '1' or '-', i.e. the 'don't care' state. Thus ti can be used to represent a group of vectors {d}, e.g. if ti=(10-1-), it represents the set {10010, 10110, 10011, 10111}. Vectors such

as ti will be used to represent clusters of rules satisfying a certain similarity criterion. This criterion is discussed next.

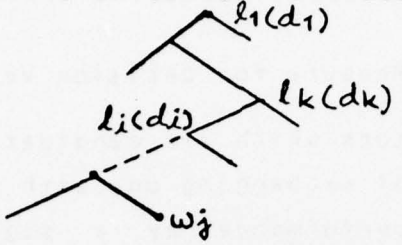


Figure 5.3.

Consider the decision tree shown above in Fig. 5.3. If di denotes a decision vector used at node li,then under the assumption that the features used at the various nodes are statistically independent, the average correct recognition rate of the tree, Pc(T), can be written as a sum of products, as:

$$Pc(T) = \sum_{k=1}^{M} P(w_k) \cdot \prod_{\ell_i \in S(\omega_k)} Pc(w_k/d_i)$$

To compare the similarity in the tree performance Pc(T), when two alternative rules d, and d' are used at some node, li, one can define an M dimensional vector function, fi(d), which maps any decison rule, d, used at li, into a point in M-space, given by :

$$fi(d) = \begin{bmatrix} Pc(w1/d) \\ Pc(w2/d) \\ \cdots \\ \cdots \\ Pc(wj/d) \\ \cdots \\ Pc(wM/d) \end{bmatrix}$$

where, it is assumed that w1,w2...wM, are the M classes below node li, in the tree.

A similarity criterion between the two decision rules, d and d', could then be defined as the l1-norm,

$$\|fi(d)-fi(d')\| = \sum_{k=1}^{M} |Pc(w_k/d)-Pc(w_k/d')|$$

The change in Pc(T), denoted by $\Delta Pc(T)$, when d is replaced by d' at node li, all other decision rules remaining unaltered, is,

$$\Delta Pc(T) = \sum_{j=1}^{M} \Big\{ P(wj) \cdot \prod_{\substack{l_k \in S(w_j) \\ l_k \neq l_i}} Pc(wj/dk) \Big\} \cdot (Pc(wj/d) - Pc(wj/d'))$$

Since $Pc(wj/dk) < 1.$, it follows that,

$$\Delta Pc(T) \leq \sum_{j=1}^{M} P(wj) \cdot \{Pc(wj/d) - Pc(wj/d')\}$$

$$\leq \| fi(d) - fi(d') \| \cdot \sum_{j=1}^{M} P(wj)$$

Hence, the smaller the distance between the vectors d and d', as measured by the l1-norm, the smaller the change in Pc(T), when rule d' replaces d.

Other choices for the norm are the Euclidean norm or the infinity norm. However, for the rest of this discussion, the l1-norm is assumed to be the distance measure used.

Using the norm defined above, the clustering problem is that of splitting the set of rules, D, into disjoint sets, C(1), C(2),...C(N), such that for any set C(i),

(a) If d, d' are in C(i),
$\| f(d) - f(d') \| \leq r$, $r > 0$.
Here, r is the spread of the cluster.

(b) any vector, $d \in D$, is in some cluster,
C(i), i=1,2,...N.

One method of performing this clustering, is to compute f(d) for all 2^m vectors in D. These M-dimensional vectors can then be grouped together in M-space using the criterion

(a). However, such a method requires the computation of M. $(2)^m$ functions, such as Pc(wk/d), in addition to the large cost of storing the vectors themselves.

5.1.2.2. Clustering Vectors in M-space

This section describes a simple method of clustering the decision vectors at any node using the similarity measure defined above. The resulting clusters are optimal in the sense that each cluster can be described with a single prototype vector with the largest number of 'don't care' bits, '-'. One would like to partition the total set of decisons, D, into sets, C(k), k=1,2,....N, such that,

(1) Any vector, d falls in at least one set C(k).
(2) For any two vectors d, d' in C(k), the distance,

$$\|f(d)-f(d')\| \leq RD, \quad RD > 0.$$

where,

$$f(d)=[\ Pc(w1/d)\ Pc(w2/d) \quad .... \quad Pc(wM/d)\ ]'$$

and $\|\cdot\|$ denotes the l1-norm introduced earlier.

Unlike data clustering in feature space, however, the criterion sought to be optimized here, is slightly different. In data clustering, the criterion maximized is usually some ratio of between cluster scatter to within-cluster scatter, since the goal is to partition the data into disjoint sets whose members are similar within a group and sufficiently different from those in other groups. In the decision clustering task, our goal is to minimize the NUMBER of clusters, where, members of a cluster satisy property (2) above. The reason for this is that the resulting set prototypes will be used in the DP procedure and the computations needed in that algorithm increase exponentially with the number of vectors(prototypes) to be

considered.

Consider the following simple method of generating the sets, C(k), and their prototypes, denoted by tk. For each bit, i, i=1,2...m, compute the figure of merit, $\delta(i)$

$$\delta(i) = \sum_{j=1}^{m} pj(i)$$

where the sum is over all classes, j=1,2,...M.

Order the m bits using their figure of merit, and select the 'don't care' bits of the prototypes to be the n lowest merit bits such that,

$$RO > \sum_{i=1}^{n} \delta(ki) \qquad \text{.....(a)}$$

where the ki are such that,

$$\delta(k1) \leq \delta(k2) \leq \delta(k3) \leq \text{.....} \leq \delta(km)$$

and n is the largest integer such that the equ. (a) holds.

The prototypes, tk, are those obtained by inserting all possible combinations of 0/1 bits in the m-n bit positions which are not 'don't care' positions. Hence the number of sets, C(k) created are,

$$N = 2^{m-n}$$

Searching over the prototypes rather than the original vectors,d, has thus resulted in a decrease in computation by a factor of 2^n . The next section defines the concept of an efficient decision rule, which allows one discard many of these sets C(k) and thus effect a further reduction in the computational burden of the DP algorithm.

5.1.3. Efficient Decision Strategies

Even after the clustering of decision rules, $d \in D$, applicable at a tree node, one may still have a large set of

possibilities to examine. The concept of an 'efficient' decision rule allows one to reject a large number of rules, because it can be shown that the optimal rules for the tree nodes (i.e. those to maximize the tree performance) must also be efficient. Since in many applications, the set of efficient rules is significantly smaller than the set of all rules, this property of the optimal decision policy can be used to reduce the computations required in the dynamic programming algorithm of Sec.5.1.1 for finding the optimal 'one-step' (OS) policy.

Before proceeding to describe a method of obtaining the set of efficient rules for each tree node, some definitions are needed.

Definition: Let di and di' be two distinct decision rules used at a tree node, li. Let f(di) denote the vector function of correct recognition rates for the M classes below li, when rule di is used at the node, i.e.

$$f(di) = \begin{bmatrix} Pc(w1/di) \\ Pc(w2/di) \\ \cdot \\ \cdot \\ Pc(wM/di) \end{bmatrix} \quad \text{assuming } W(li) = \{w1, .. wM\} .$$

Then rule di dominates di' if f(di) > f(di'), i.e.

$Pc(wj/di) \geq Pc(wj/di')$ for all wj W(li)
and strict inequality holds for at least one class wj.

Definition: A decision rule, di, is efficient at node li, if there is no other rule di' which dominates it.

The following theorem establishes the fact that the optimal decision rules at the tree nodes must be efficient.

Theorem 1: Let {li}, i=1,2,....N, denote the nodes of a tree. Let Di, for i=1,2,..N, be the set of rules, di, applicable at the corresponding nodes, li. Then the set of optimal rules, di*, i=1,2..N, which maximize the tree recognition rate must be efficient at the corresponding nodes, li, i=1,2,..N.

Proof: Under the assumption of statistically independent node decisions, the tree performance, Pc(T) is given by,

$$Pc(T) = \sum_{\omega_i} P(w_i) \cdot \prod_{l_k \in S(\omega_i)} Pc(w_i/d_k)$$

Consider some particular node, say dk and assume that dk is not efficient at lk. Then there exists some other rule, say dk' such that,

$Pc(w_i/d_k') \geq Pc(w_i/d_k)$ for all $w_i \in W(l_k)$, and strict inequality holds for at least one class wj in W(lk). Hence if all other decision rules are kept fixed and dk is replaced by dk' then the value of Pc(T) can only increase since in the equation for Pc(T) given above, the sum of the product terms for classes under node lk will increase. Hence dk can never be an optimal rule at lk, because exchanging dk for dk' would increase the tree performance. Since the choice of the node lk was arbitrary, the same argument holds for any other node, and hence it follows that the optimal rule at every node must also be an efficient rule.

The preceding definitions and theorem hold for any type of decision rule, whether it uses discrete or continuous features to make the branching decision at the node.

However, the remaining discussion in this section will be concerned with the case of discrete non-metric features, each possessing m states, since in such cases the number of rules in any set, Di, becomes large for large m, eg. $|Di| = 2^m$, for a binary decision tree. One can always find the set of efficient rules at any node, li, by computing the function, f(di) for all di$\in$ Di, and finding the rules which are not dominated by any other rules. However, for the discrete feature case, this 'brute-force' method is impractical for large m. In this section , we propose the use of a branch-and -bound method for finding the set of efficient rules.

This method consists of sequentially assigning bits of a prototype vector, t, the values 0 or 1, and at each step, computing a lower bound and upper bound on the vector function, f(d) for all vectors d, represented by that prototype. Whenever the upper bound function becomes less than the function f(d*) for some known efficient rule, d*, the entire set of rules represented by the prototype, t, can be discarded as not being efficient, since they would be dominated by d*. The lower bounding function is useful for finding a group of rules that is guaranteed to contain a rule d, which is NOT dominated by d*. This is done by checking if any component of the lower bounding function for t has a value greater than the corresponding component of f(d*).If this is the case, it follows that none of the rules in that set can be dominated by d*.Hence members of that set are potential candidates for inclusion in the set of efficient strategies. Whenever a set of rules is discarded as being not efficient, the algorithm backtracks to the previous stage and tries a new bit assignment to the prototype, that it has not yet tried. The algorithm terminates when it has found all sets of rules that are

dominated by a given rule, d*. During its execution it also keeps track of potential sets that contain efficient rules. These rules can be then used to discard more rules not already discarded. The algorithm is formally described below for the case of a binary decision node, though it can be readily extended for use at a M-way decision node, where M>2.

The lower and upper bounds on the vector function f(d) for rules in the cluster, D(t), are obtained in the following way. Let N0(d), and N1(d) be the set of indices, $j \in \{1,2,\ldots m\}$, such that dj=0 or 1 respectively. These represent the states of the observed feature for which the decision is to go left/right at that node. If rule d is used at node l, and W0(l) and W1(l) are the sets of classes distinguished at l, then for any class $wi \in W0 \cup W1$, the correct recognition rate at that node is,

$$Pc(wi/d) = \sum_{j \in N0(d)} pi(j) \quad \text{if } wi \in Wo(l),$$

$$= \sum_{j \in N1(d)} pi(j) \quad \text{if } wi \in W1(l).$$

Bounds on Pc(wi/d) for rules $d \in D(t)$ can then be obtained as,

$$\text{Max}_{d \in D(t)} Pc(wi/d) \leq \begin{cases} 1 - \sum_{j \in N_1(t)} pi(j) & \text{if } wi \in Wo(l), \\ 1 - \sum_{j \in N_0(t)} pi(j) & \text{if } wi \in W1(l). \end{cases}$$

$$\text{Min}_{d \in D(t)} Pc(wi/d) \geq \sum_{j \in N_0(t)} pi(j) \quad \text{if } wi \in W0(l),$$

$$\sum_{j \in N_1(t)} p_i(j) \quad \text{if } w_i \in W1(l) \quad . \quad ..(5.3)$$

Using equations (5.3) which bound each component of the vector function f(d), the bounds on the vector f(d) can be defined as,

$$\min_{d \in D(t)} f(d) \geq l(t) = [\, l1(t)\ l2(t)\ \ldots\ lM(t)\,] \text{ where,}$$

$$l_i(t) = \sum_{j \in N_0(t)} p_i(j) \quad \text{if } w_i \in W0(l)\ ,$$

$$\sum_{j \in N_1(t)} p_i(j) \quad \text{if } w_i \in W1(l)\ .$$

$$\max_{d \in D(t)} f(d) \leq u(t) = [\, u1(t)\ u2(t)\ \ldots\ uM(t)\,] \text{ where,}$$

$$u_i(t) = 1 - \sum_{j \in N_1(t)} p_i(j) \text{ if } w_i \in W0(l)\ ,$$

$$1 - \sum_{j \in N_0(t)} p_i(j) \text{ if } w_i \in W1(l)\ .$$

The branch-and bound algorithm which uses these vector bounds to detect all rules dominated by a given rule, d*, is descibed next.

Algorithm :

Variables used: d* is a rule assumed to be efficient at the start of the algorithm. The program finds all rules dominated by d* and adds them to a list, RL of 'rejected rules'. It also finds all rules NOT dominated by d* and puts them in list EL of possible efficient rules for consideration later. t denotes a prototype initialized to all '-' bits. at the start. l(t) and u(t) are bounds on f(d) for rules in D(t) . The bits of t are set to 0 or 1 in a fixed order. At stage k, the k th. bit of t is being changed. The program uses m flags, ci, i=1..m to keep track of the bit assignments of t that it has tried.

Initialize:Set t=(----..-), l(t)=(0,0,..0), u(t)=(1,1,..1), RL = EL = ϕ , the empty set, and stage variable, k=1 .

Step 1:(Set k th. bit of t) If k=0 STOP, else, set ck=ck+1 ; if ck > 2 go to step 4 ; if ck=2 set tk=1, if ck=1 set tk=0.Compute l(t) and u(t) and continue.

Step 2: (Check if rule t is to be discarded) If f(d*) > u(t), put t in RL and go to step 1, else continue.

Step 3: (Check if t dominates d*) If l(t) > f(d*) , then put t in EL, replace d* by any rule d' $\in$ D(t) and go to step 1; else check if there is some component li(t) such that li(t) > fi(d*), and if there is one, put t in EL and go to step 1;else if no such component is found, go to step 5.

Step 4: (Backtrack to previous stage) Set ck=0, tk='-', and k=k-1 and go to step 1 .

Step 5: (Increment stage variable and check for last stage) If k=m put t in EL and go to step 1; else set k=k+1 and go to step 1.

Example 1 illustrates the execution sequence of the algorithm. The class-conditional probabilities of the feature (5 states) are tabulated below for each of the 4 classes. The classes {w1,w2} are to distinguished from {w3,w4} using this feature. The algorithm is used to find all rules dominated by d*=(01110) and those prototypes that could contain rules not dominated by it. Fig. 5.4 shows the search tree expanded in the order indicated by the circled numbers. The vector t, and its bounds, l(t) and u(t) are shown at some nodes. The circled terminal nodes indicate rejected rules, while those put in list EL for examination later, are marked by '*'.

Example 1:

The class-conditional probabilities of the feature are shown in the table below.

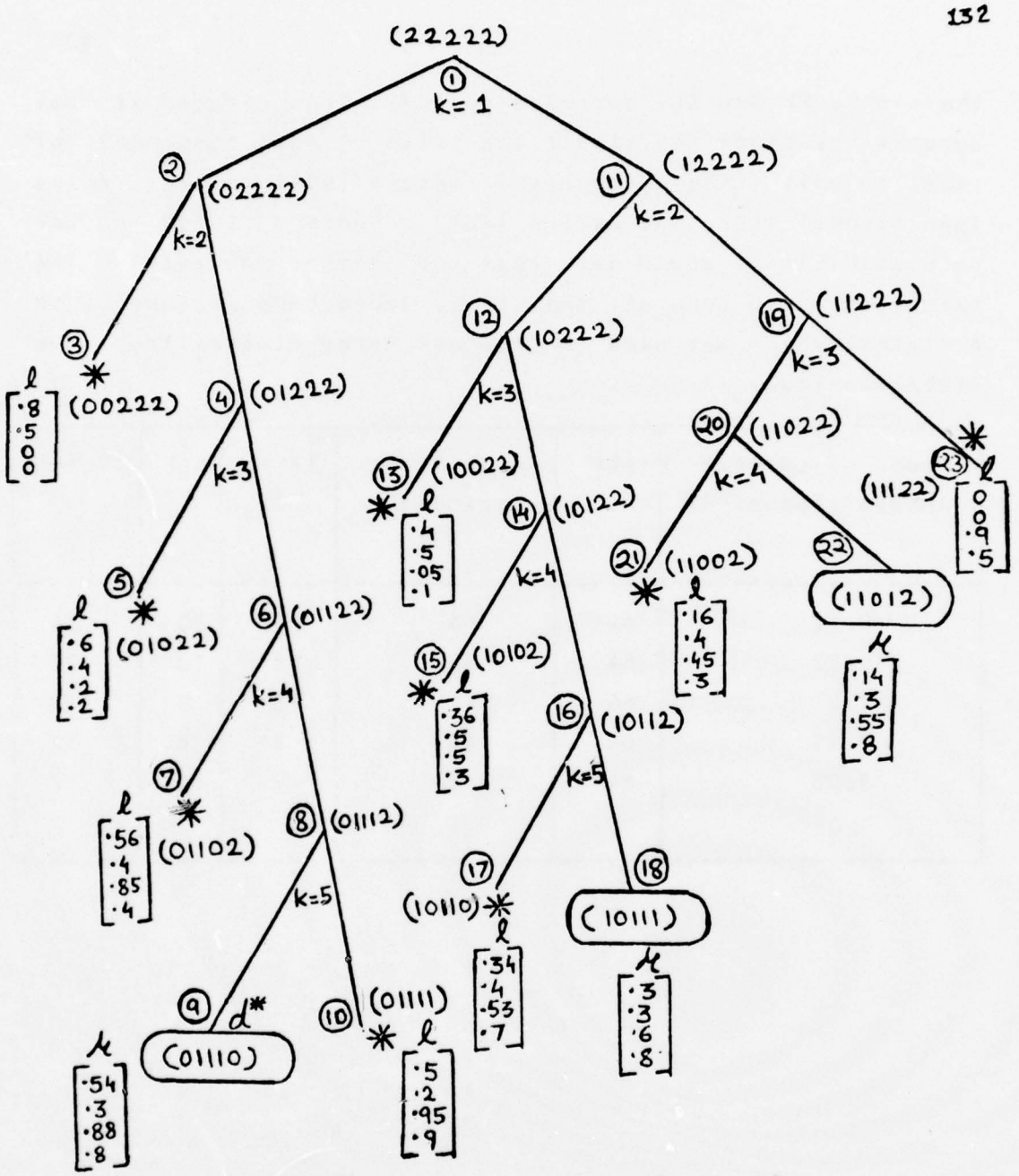
Class	pi(1)	pi(2)	pi(3)	pi(4)	pi(5)
1	0.5	0.3	0.1	.06	.04
2	0.2	0.3	0.2	0.2	0.1
3	.05	0.4	.45	.03	.07
4	0.1	0.2	0.2	0.4	0.1

Example 2:

The following example depicts the variation in the algorithm's search efficiency (measured here by the number of nodes expanded) as a function of the average performance of the rule, d*. The table shows that as the rule d* becomes 'perfect' (i.e. error rate=0), the number of nodes expanded in the search tree as well as the sum of prototypes added to

the lists, RL and EL, decrease rapidly. This result is not surprising, since the closer the value of each component of f(d*) to unity, the quicker the search will detect rules (prototypes) such that u(t) < f(d*) . Hence a large number of possibilities would get rejected rather quickly. The example used to generate this table employed a feature with 6 states, which was used to separate three classes from two other classes.

Prob. of Correct Recognition of d*	|D|	Total Nodes Expanded	|RL|	|EL|	|RL+EL|
0.72	64	66	14	20	34
0.82	64	52	17	10	27
0.87	64	42	13	9	22
0.90	64	26	8	6	14
1.00	64	2	2	0	2



$d^* = (01110)$ $\quad W_0 = \{\omega_1, \omega_2\}$

$f(d^*) = \begin{bmatrix} .54 \\ .3 \\ .88 \\ .8 \end{bmatrix}$ $\quad W_1 = \{\omega_3, \omega_4\}$

Search Tree For Example 1.

Fig 5.4

5.2. Feature Selection at Tree Nodes

A method for optimally selecting the order of feature measurements along every tree path, was outlined in Chapter 4. However, that method involves a very high computational and storage cost, even for moderately sized feature sets and feature states. For example, if there are N features, each taking on m states, then for a node n levels deep from the root, the optimal feature to be measured must be tabulated for all possible n feature subsets, and all possible measurements of these features. This requires, approximately,

$$\binom{N}{n}\binom{n}{(m)} \quad \text{words.}$$

To avoid this storage cost, one could take the opposite approach of not taking into account the hisory of measurements leading to a node,lk, but rather choose the feature at a node on the basis of maximum node performance, Pc(lk). It was shown in Chapter 3 that this way of selecting features, does not necessarily result in the optimal tree performance. Analogous to the discussion on decision policy design, one seeks to steer a middle course, viz., choosing the feature at a node taking into account the features used above and below it in the tree, while not taking int account the actual measurements taken on the sample before it arrived at lk. Two methods for obtaining the optimal (for statistically independent features) solution to this problem are given in the next two sections.

5.2.1. A Dynamic Programming Formulation

The bottom-up approach of dynamic programming can be used to select the optimal feature to be used at each tree node.Starting one level above the terminals, one computes the best feature to be used at that node for all possible feature assignments at nodes leading to that node.The best feature choice is made on the assumption that a maximum likelihood rule is used at each node (OSMLR) on this path, and that the node decisions are statistically independent. For a node n levels deep, one must therefore, evaluate,

$\binom{N}{n}\, n!$ if there are N features .

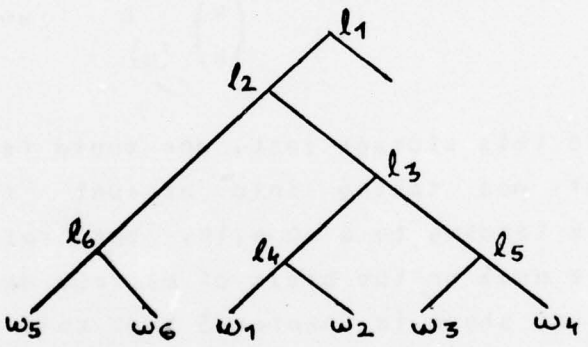


Fig. 5.5

Consider the tree of Fig. 5.5, and let, Pc(wi/lk,f) represent the correct recognition rate of a sample from wi at node lk, if feature f is used there and a maximum-likelihood policy employed.The total tree performance can be written as a sum of products of the Pc()'s (see equ. (3.1)), and the contribution to this performance by the samples that filter down the path, l1,l2,l3,l4, is given by,

$$C(l1,l2,l3,l4) = \sum_{i=1}^{2} P(wi).Pc(wi/l2,fl2)Pc(wi/l2,fl2)$$
$$Pc(wi/l3,fl3)Pc(wi/l4,fl4)$$

Thus, one can define the optimal feature to be measured at l4, f4* as a function of features, fl1,fl2,fl3,as f4*(fl1,fl2,fl3), which maximizes C(l1,l2,l3,l4), when fl1,fl2,fl3 are used at l1,l2,l3, respectively. Similarly, f5*(fl1,fl2,fl3) can be computed. Working up one level in the tree, the optimal feature, f3*(fl1,fl2) is computed as that which maximizes the contribution of the path, l1,l2,l3, viz C(l1,l2,l3), when fl1,fl2,fl3, are used at l1,l2,l3, and f4*(fl1,fl2,fl3), and f5*(fl1,fl2,fl3) are used at l4,l5. In this way f1* is computed as the best feature to measure first. By table look-up in the tables, f2*(fl1), f3*(fl1,fl2), etc. successively, all the features can be determined.The tables f4*(), f5*() are optimal, since all possible features are considered at these nodes, for a given feature useage on the path l1,l2,l3. The optimality of the above algorithm then follows by induction on successively higher level tree nodes, and from the fact that the contribution of each subtree below a node, lk, to the goodness measure, c(l1,...lk) is additive. Since the maximum value of C(l1) is the optimal tree performance, the corresponding feature assignment to the nodes is also optimal.

5.2.2. A Branch and Bound Formulation

In this section, we describe an alternative method for obtaining the optimal feature assignment for a given tree skeleton. This procedure falls in the category of branch-and-bound methods, which have found wide use in many combinatorial optimization problems. To the best of our knowledge, the use of this method for selecting the optimal features for a tree skeleton is new.

The branch and bound method assigns features to the tree nodes in a sequential(top-down) manner. Whenever the assignment of a feature reduces the optimality criterion (e.g. correct recognition rate) below a lower bound, that assignment sequence is abandoned. The algorithm backtracks to the previous stage(node) and tries a new sequence. At the conclusion, the resulting sequence is the optimal feature assignment. The following notation is used in the discussion below:

F=total set of N features.

flk=feature assigned to node lk.

Pc(wi/flk)=prob. of correctly classifying a sample from wi into W0(lk) (or W1(lk)), using flk, and a maximum likelihood rule.

Ln=(l1,l2..ln)=an ordering of the nodes, such that, if lj precedes li on a path from the root to a terminal then, j < i .

Lm=(l1,l2,..lm)=a prefix of length m of Ln, representing a subgraph Tm of tree , T.

Pc(fl1,fl2,..flm)=correct recognition rate of Tm, using, fl1,fl2,,,, at the m nodes of the subtree, Tm.This tree classifies a sample into m+1 sets. If m=n, the sets are single classes, else they are unions of classes.

Pc*(m)= max recognition rate of Tm

$$= \max_{\forall f_{\ell_1}, f_{\ell_2}, \ldots f_{\ell_m}} Pc(fl1, fl2, \ldots flm)$$

$$= \max_{\forall f_{\ell_1}, \ldots f_{\ell_m}} \sum_{\omega_i} [P(wi) \cdot \prod_{l_k \in S(\omega_i) \cap Lm} Pc(wi/flk)]$$

Example:

The figure below shows the total tree, Tn, for n=5, and the partial trees, T2, and T3, assuming that Ln, the linear ordering of the nodes, is chosen to be l1,l2,l3,l4,l5.

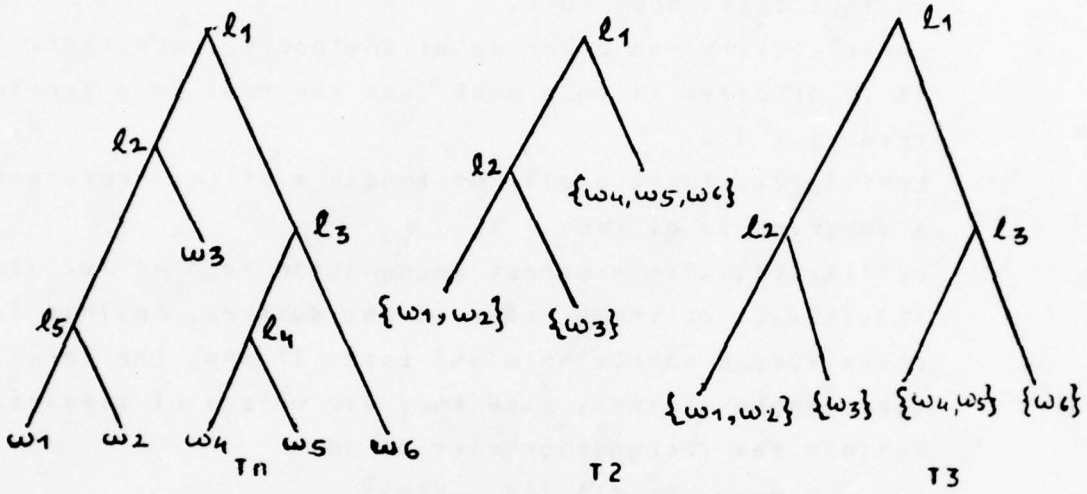


Fig. 5.6

It follows from the fact that Pc(wi/flk) < 1. that,

(i) $Pc(fl1, fl2, \ldots flm) \geq Pc(fl1, fl2, \ldots flm, flm+1)$

(ii) $1. \geq Pc^*(1) \geq Pc^*(2) \ldots\ldots Pc^*(n)$

Branch-and Bound Algorithm.

Variables used: tij= array of flags, in which tij=1 denotes that fi has been assigned to node, lj.
B=lower bound on the optimal recognition rate of T, Pc*(n) .

Step 0(Initialize): Set tij=0, i=1,2,..m, j=1,2,..n
Choose an initial assignment of features, and set B to the recognition rate for that choice.
Set k=0 (stage=0), and L0=(), the null sequence.

Step 1(choose next feature): Compute Lk from Lk-1, as Lk=(Lk-1,fa), where, $Pc(Lk-1,fa)=\max_{f \notin L_{k-1}} Pc(Lk-1,f)$.

Step 2(Test against lower bound):
If Pc(Lk-1,fa) < B go to step 4 else go to step 3.

Step 3 (Check for last stage): If k=n go to step 6 else set tak=1, k=k+1 , and go to step 1.

Step 4(Backtrack): Set tjk=0, j=1,2...m, k=k-1.
If k=0 STOP, else go to step 5.

Step 5(Seek new feature at previous stage):
If tjk=1, j=1,2,..m go to step 4 else go to step 1.

Step 6(Update lower bound):
Set B=Pc(Ln), and record Ln, the optimum assignment so far then go to step 4.

The above algorithm satisfies the following properties.

Theorem 1: Let Pc*(n) be the optimal recognition rate achievable. Then, the algorithm evaluates every sequence, fl1,fl2,..flk, which has the property that,

$$Pc(fl1,fl2,..flk) \geq Pc^*(n), \text{ for } 1 \leq k \leq n .$$

Proof: $Pc(fl1,fl2,..flk) \geq Pc^*(n) \geq B$, for any value of B that occurs in the execution of the algorithm. Hence, this feature sequence will be evaluated, since, the algorithm evaluates all sequences whose Pc value exceeds the lower bound ,B, at that time.

From Theorem 1, one obtains,

Theorem 2: If in the algorithm, we keep track of the maximum value of Pc(k) at the k th. stage, then at the conclusion of the algorithm, this value is equal to Pc*(k), k=1,2...n. Hence, one also obtains the solution to the optimal assignment problem for partial classification trees {Tm), m=1,2,..n.

Proof: From property (i) stated earlier,

$Pc^*(k) \geq Pc^*(n)$ for $k \leq n$. Hence,

$Pc^*(k) \geq B$, any lower bound on Pc*(n).

At the k th. stage, we examine all sequences Lk such that $Pc(Lk) \geq B$, and hence, ALL Lk such that $Pc(Lk) > Pc^*(n)$. If Lk* is the optimal k-sequence, i.e. $Pc(Lk^*)=Pc^*(k)$, then this sequence also satisfies $Pc(Lk^*) \geq Pc^*(n) \geq B$. Hence, it will be evaluated, and by keeping track of the maximum value of Pc(), we obtain Pc*(k) when the algorithm terminates.

5.2.3. Feature Ranking

The last two sections proposed alterntative methods of obatining the optimal feature assignment for a given tree skeleton under the assumption that an OSMLR rule is used at each node. The number of features to be considered at a particular node during this search for the optimal assignment, can often be reduced by feature ranking. For each feature in the total set of features, one could compute the vector of correct recognition rates for the classes under a given node. Thus one can define a goodness measure, g(f,li) for all feaures f∈ F at node li, as,

$$g(f,li) = [\ Pc(w1/li,f)\ Pc(w2/li,f)\ \ldots\ldots\ldots]$$

where w1, w2,... are classes in W(li), and each component of the vector measures the probability of correct decision on a sample from that class at that node, using feature f and an OSMLR rule. Analogous to the concept of efficient decision rules (Sec.5.1.3), one can then define the notion of dominance among features and that of efficient features at each node.

Definition: A feature f dominates feature f′ at node li if $g(f,li) > g(f',li)$, i.e.
$Pc(wj/f,li) \geq Pc(wj/f',li)$ for all $wj \in W(li)$ with strict inequality holding for at least one class, wj.

Definition: A feature f is efficient at node li if it is not dominated by any other feature, $f' \neq f$ at that node.

It can be easily shown that the optimal features for a given tree skeleton must be efficient. The proof is identical to that employed for efficient decision rules in Sec.5.1.3 and is therefore omitted here.

6. Estimation Of Decision Rules From Finite Samples

In the previous chapters, we have discussed optimization methods for the design of hierarchical classifiers, when the criterion of optimality is a weighted sum of measurement cost and misclassification risk. These methods assumed that the required parameters such as the class-conditional probabilities of the features used , were either known or could be estimated with a high degree of confidence from training samples. The classifier design was therefore not influenced by the number of samples in the design set. However, when the sample size is 'small', the degree of confidence in the estimated parameters is low, and one would like to use the samples as efficiently as possible. A classification scheme that requires too large a set of parameters to be estimated, could in such cases perform worse on a independent test set than a scheme which requires fewer parameters.

The number of parameters required, is a function of the number of features used at each node, the number of states(levels) of each feature (if the features are discrete), and the decision complexity at each node. Decision complexity refers to the number of sets (of classes) distinguished at a node (e.g. this is two for a binary tree). As an example, consider the following two tree structures for a 6-class problem.

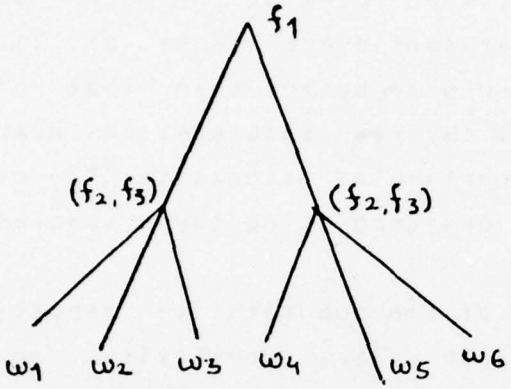


Fig. 6.1a

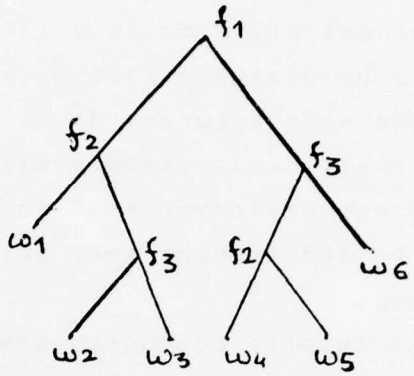


Fig. 6.1b

Assume that at each tree node, a maximum likelihood rule is used to distinguish the sets of classes. In T1, the top node uses feature f1 to separate the two mixtures, {w1,w2,w3} and {w4,w5,w6}, while at its lower level nodes, there is a 3-way decision made using the observations of features f2 and f3. In T2 each node decision is binary. Assuming that each feature has m states and the features are class- conditionally statistically independent, the number of parameters required for trees T1 and T2 are,

$$Np(T1) = 2.m + 2.3.2.m = 14.m$$
$$Np(T2) = 2.m + 2.2.m + 2.2.m = 10.m$$

The tree, T2, therefore requires fewer parameters than T1. In theory (i.e. assuming perfect knowledge of probabilities) T1 should perform better than T2, since the classes {w1,w2,w3} are being separated in T1 using features f2 and f3, while in T2 this subproblem is solved in two stages, and the discrimination between w2 and w3 makes no use of information in f2. However, since T2 requires fewer parameters than T1, for small sample sizes, one can expect

that the estimated maximum likelihood rules at the nodes of T2 could be designed with greater confidence those of T1. Hence the mean accuracy of T2 could be better than that of T1 for small design sample size. By mean accuracy, we mean the average performance of the particular classifier(T1 or T2) on an independent test set, designed using say N samples per class.

This chapter discusses some of the quantitative aspects of the relationship between sample size, complexity, and classifier performance, for the case when the features are discrete and class-conditionally statistically independent, and under the assumption that a maximum likelihood rule is used at each tree node of a hierarchical classifier. This analysis is carried out in the following phases.

First, it is shown that for 'most' probability structures, the variance of the estimated parameters of a mixture of M populations (classes) is less than the sum of the variances of the estimates of the individual class probability functions, assuming that the number of samples available is proportional to the apriori probability of the classes. This result provides a heuristic argument for stating that the confidence in estimating a maximum likelihood rule for splitting a set of classes into two sets, would be greater than the confidence in estimating a rule that, in a single step, splits a set of classes into the component classes.

We then derive an expression for the mean accuracy of a decision tree used for an M-class problem. The mean accuracy is defined as the performance of an estimated (using a finite sample) hierarchical classifier, averaged over the 'space of all problems'. It is assumed that all problems are equally likely. Though the resulting expression is too general to enable any precise design criteria to be derived,

it provides a result which is qualitatively significant. It is shown that for a given sample size, there exists an optimal quantization complexity, i.e. by increasing the number of quantization levels incefinitely, one cannot achieve increasingly better performance. While this result has been reported by others [33,34], our analysis extends the result to a multiclass (more thar two classes) problem.

The above result establishes a relationship between optimal quantization complexity and sample size. The last section in this chapter investigates the relation between decision complexity and sample size. It shows that for small sample sizes, it may be better to split a multiclass problem hierarchically into simpler problems (i.e. into problems with a smaller complexity) than to design a single-step classifier which uses all features together to make an M-way decision.

6.1. Estimation Of Discrete Probabilities Of Mixtures

Assume that one wants to estimate the probability distributions of samples from M classes, wi, i=1,2,....M. For simplicity, each sample is regarded as a single feature observation, where the feature can have one of m values(states). Let pi(j), for j=1,...m, i=1,2,..M, denote the probability that the feature will be in state j given that it came from class wi. The maximum likeihood estimate of pi(j), written as $\hat{p}i(j)$, is given by,

$$\hat{p}i(j) = ni(j) / Ni$$

where, Ni are the number of labelled design samples from wi, and ni(j) are the number of samples in it that were in state j. Assuming that the samples were independent and identically distributed, the marginal probability of obtaining a particular value of ni(j) is given by a binomial

distribution, viz.,

$$Pr[ni(j)] = B(Ni, ni(j), pi(j))$$

$$\binom{Ni}{ni(j)} [pi(j)]^{n_i(j)} \cdot [1-pi(j)]^{N_i - n_i(j)}$$

The mean and variance of ni(j) and $\hat{p}i(j)$ are given by,

$$E[ni(j)] = Ni.pi(j) \qquad E[\hat{p}i(j)] = pi(j)$$
$$Var[ni(j)] = Ni.pi(j).[1-pi(j)]$$
$$Var[\hat{p}i(j)] = pi(j).[1-pi(j)] \,/\, Ni \qquad .....(6.1)$$

Consider the problem of estimating the state probabilities assuming a sample came from the mixture of populations of w1,..wM, with known mixing weights, P(wi), i=1,2,..M. Let p(j) denote the probability of observing state j. There are two ways of estimating the parameters, p(j), j=1,2...m.

In the first scheme, assume that one could choose Ni independent samples from each of the populations, wi, i=1,2..M. Then a maximum likelihood estimate of p(j) is given by,

$$\hat{p}(j) = P(wi).\, ni(j) \,/Ni$$

where ni(j) has the connotation defined earlier.
From equation (6.1), it follows that,

$$E[\hat{p}(j)] = P(wi).E[\hat{p}i(j)]$$

$$Var[\hat{p}i(j)] = \sum_{i=1}^{M} P(wi)^2 \cdot Var[\hat{p}i(j)]$$

$$< \sum_{i=1}^{M} Var[\,\hat{p}i(j)\,] \text{ since } P(wi) \leq 1.0$$

For the case when all classes have the same apriori

probability, P(wi), the above inequality reduces to,

$$\mathrm{Var}[\hat{p}(j)] < 1/M \cdot \max_{1 \le i \le M} \{\mathrm{Var}[\hat{p}i(j)]\} \quad \ldots\ldots\ldots\ldots (6.2)$$

The above analysis shows that the variance of the mixture parameters in such an estimation scheme, is less than the sum of individual class parameter variances, and for equal apriori class probabilities, it is less than 1/ M times the maximum variance of the class parameters.

An alternative method of estimating p(j) might consist of taking N independent samples from the mixture population. The probability that in a random sample so selected, the feature will be in state j, is given by,

$$p(j) = \sum_{i=1}^{M} P(wi) \cdot pi(j) \quad \ldots\ldots\ldots\ldots\ldots\ldots (6.3)$$

If n(j) denotes the number of samples out of N that were in state j, then a maximum likelihood estimate of p(j) is,

$\hat{p}(j) = n(j) / N$.

Since each sample has a probability p(j) of being in state j, the marginal distribution of n(j) is binomial, and given by,

$Pr[n(j)] = B(N, n(j), p(j)]$, where p(j) is given by (6.3).

For the case of equal apriori class probabilities, P(wi), the variance and expectation of $\hat{p}(j)$ are,

$$E[\hat{p}(j)] = p(j) = 1/M \cdot \sum_{i=1}^{M} pi(j)$$

$$\mathrm{Var}[\hat{p}(j)] = p(j) \cdot [1-p(j)] / N$$

$$= \frac{1}{M^2} \frac{\sum_i pi(j) \left[M - \sum_i pi(j)\right]}{N}$$

Denoting by xi the quantity pi(j), it follows fom the above equation and (6.1), that,

$$\mathrm{Var}[\hat{p}(j)] \leq \sum_{i=1}^{M} \mathrm{Var}[pi(j)] \quad ,$$

assuming that N/M samples were available to estimate the pi(j), if the following condition holds:

$$1/M^2 \cdot \left(\sum_j xj\right) \cdot \left(M - \sum_j xj\right) \leq M \cdot \sum_j xj \cdot (1-xj)$$

The above inequality reduces to,

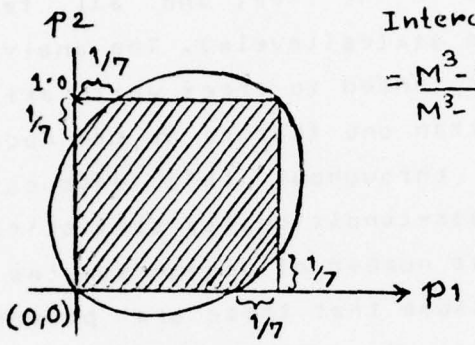
$$(M^3 - M) \cdot \sum_j xj + \sum_k \sum_j xj \cdot xk - (M^3 - 1) \cdot \sum_j (xj)^2 < 0. \qquad (6.4)$$

Equation (6.4) is an ellipsoidal surface in (M-1) dimensional space that passes through the origin and the point, (1,1,..1), and has an intercept on each axis, xj, given by,

$$xj = (M^3 - M) / (M^3 - 1) \quad .$$

Each point in this space represents a set of class probabilities, pi(j), i=1,2,....M, and all situations where the point falls within the ellipsoid represents a case where the mixture parameter has a variance less than the sum of the class parameter variances. For a large number of classes, M, the intercept on each axis tends to unity, and hence most of the hypercube in which the probabilities, pi(j) fall, lies within the ellipsoid, and the previous statement is true. Hence, one can state that for 'most' situations, the mixture parameter can be estimated with a smaller variance, than the sum of variances of the

individual class parameters. For the case of two classes, M=2, the ellipse obtained from (6.4) is depicted in the figure below. The shaded area shows the region in which the inequality (6.4) is valid.



Intercept on axes

$$= \frac{M^3 - M}{M^3 - 1} = \frac{8-2}{8-1} = \frac{6}{7}$$

for M=2

Fig. 6.2

6.2. Mean Accuracy Of A Hierarchical Classifier

The mean accuracy of a classifier is defined[33,34] as the performance of a classifier averaged over the space of all problems. A problem consists of the true set of parameters defining the class distributions. In this disscussion, we assume that a hierarchical classifier is used to distinguish between M classes, using discrete features, whose probability functions are, $(p_k^i(j))$, where i refers to the feature number, k is the class, and j rangess over the states of the feature. For simplicity, it is assumed that the parameters, $(p_k^{\hat{}}(j))$ are uniformly distributed, i.e. all problems are equally likely.

For the purposes of this discussion, we assume that a binary balanced decision tree is used for an M-class problem, where each class label occurs only at one terminal node. Hence the number of levels L, in the tree is,

$$L = \log_2 M \quad .$$

It is assumed that a single feature is measured at every nonterminal node in the tree, and all feature values are quantized into m states(levels). The analysis that follows can be readily extended to trees which are not binary, and which use more than one feature at the nodes. However, the assumption made throughout this discussion is that all features are class-conditionally statistically independent.

Let Nk be the number of design samples from class wk, k=1,2,,M, and assume that these are proportional to P(wk) the class a priori probability. Let $\{p_k^i(j), k=1,2,..M, j=1,2,..m\}$ be the class-conditional probabilities for feature i in class wk. For ease of notation, we also assume that feature i is used at node li.

Then a maximum likelihood estimate of the parameters, $p_k^i(j)$, is defined as,

$$\hat{p}_k^i(j) = n_k^i(j) \; / \; Nk$$

where $n_k^i(j)$ are the number of wk samples whose i th. feature was in state j.

The estimated maximum likelihood decision rule at li for a sample (x1,x2....xm) reaching it is,

If xi is in state j, and,

$$\sum_{w_t \in W_0(l_i)} n_t^i(j) \geq \sum_{w_t \in W_1(l_i)} n_t^i(j) ,$$

then go left below li,

else go right.

When the samples are independent, the sampling distribution of the counts, $\{n_k^i(j),\ j=1,2..m\}$ is a multinomial with m-1 degrees of freedom, given by,

$$Pr[n_k^i(j):j\in(1,m)] = \frac{(Nk)!\ \prod_{j=1}^{m}[p_k^i(j)]^{n_k^i(j)}}{\prod_{j=1}^{m}[n_k^i(j)]!} \quad \ldots\ldots(6.5)$$

Therefore, the performance of a classifier T, designed using a particular design sample set, is ,

$$Pc(T,M/\{p_k^i(j),\{n_k^i(j)\}) = \sum_{k=1}^{M} P(wk).\prod_{l_i\in S(\omega_k)}\left[\sum^{j}\delta_k^i(j).p_k^i(j)\right] \quad \ldots(6.6)$$

where,

$$\delta_{k(j)}^i = 1 \text{ if } wk\in W0(li) \text{ and } \sum_{\omega_t\in W_0(l_i)} n_t^i(j) \geq \sum_{\omega_t\in W_1(l_i)} n_t^i(j),$$

$$= 1 \text{ if } wk\in W1(li) \text{ and } \sum_{\omega_t\in W_0(l_i)} n_t^i(j) < \sum_{\omega_t\in W_1(l_i)} n_t^i(j),$$

$$= 0 \text{ otherwise.}$$

The average performance of a hierarchical classifier designed using Nk samples from class wk, k=1,2,..M, can be obtained by taking the expectation of $P_c(T,M/\{p_k^i(j)\},\{n_k^i(j)\})$ with respect to the sampling distributions given by equ. (6.5). This expectation operator will be denoted by E'. The mean accuracy of a balanced binary tree for an M-class problem, is then derived by averaging the expected performance over the 'space of all problems', using some assumed prior distribution of the parameters, $\{p_k^i(j)\}$. This expectation operator is denoted by E. The resulting expression for the mean accuracy is given by equ. (6.7). The

derivation of the result is described next.

$$\bar{P}_{cr}(T,M) = E\left\{E'[P_c(T,M \mid \{p_k^i(j)\},\{n_k^i(j)\})]\right\} \tag{6.7}$$

Assume that all values of $\{p_k^i(j),\ j \in (1,m)\}$ are equally likely, under the constraint,

$$\sum_{j=1}^{m} p_k^i(j) = 1$$

Then the probability density, $dP[\{p_k^i(j),\ j \in (1,m)\}]$, is given by [33],

$$dP[\{n_k^i(j),\ j \in (1,m)\}] = (m-1)!\ dp_k^i(1) \cdot dp_k^i(2)\ldots dp_k^i(m) \quad .$$

Denoting by $'\int'$ the integration over the range of problems, $\{p_k^i(j),\ j \in (1,m),\ k \in (1,M),\ i = 1,2\ldots\}$, and summing over all possible design samples, $\{n_k^i(j)\}$, Equ.(3) becomes,

$$\bar{P}_{cr}(T,M) = \int_{\{p_k^i(j)\}} \sum_{\{n_k^i(j)\}} \left[\sum_{k=1}^{M} P(w_k) \cdot \prod_{\ell_i \in S(w_k)} \sum_{j=1}^{m} \delta_k^i(j) \cdot p_k^i(j)\right]$$

$$\cdot \prod_{i} \prod_{t=1}^{M} P_r[n_t^i(j),\ j \in (1,m)]$$

$$\cdot \int_{i} \int_{t=1}^{M} (m-1)!\ dp_t^i(1)\ldots dp_t^i(m) \tag{6.8}$$

The product terms indexed with node labels (features), i, can be grouped together and the summation over the states, j, interchanged with the summation over the $\{ n_k^i(j) \}$ to yield,

$$P_{cr}(T,M) = \sum_{k=1}^{M} P(w_k) \prod_{\ell_i \varepsilon S(w_k)} \left[\iint \cdots \int_{\{p_t^i(j), j\varepsilon(1,m), w_t \varepsilon W(\ell_i)\}} \sum_{j=1}^{m} p_k^i(j) \cdot \sum_{\{n_t^i(j)\} \varepsilon I_k^i} \prod_{w_t \varepsilon W(\ell_i)} Pr[n_t^i(j)] \prod_{w_t \varepsilon W(\ell_i)} (m-1)!\, dp_t^i(1) \ldots dp_t^i(m) \right] \quad \ldots\ldots (6.9)$$

In the above equation I_k^i represents the set of counts $\{ n_t^i(j) \}$ such that,

$$\sum_{w_t \varepsilon W_o(\ell_i)} n_t^i(j) \geq \sum_{w_t \varepsilon W_1(\ell_i)} n_t^i(j) \qquad \text{if } w_k \varepsilon W_o(\ell_i)$$

or,

$$\sum_{w_t \varepsilon W_o(\ell_i)} n_t^i(j) < \sum_{w_t \varepsilon W_1(\ell_i)} n_t^i(j) \qquad \text{if } w_k \varepsilon W_1(\ell_i)$$

From the symmetry of the problem, it follows that each term in the summation over the states, j, must be the same. Hence, the mean recognition rate for class w_k is m times the mean recognition rate for a sample whose features are all in state 1 (say). Equ.(6.9) can therefore be written in the following way, after substituting for the probability distribution of $\{n_t^i(1),\ t \in (1,M)\}$

$$\bar{P}_{cr}(T,M) = \sum_{k=1}^{M} P(w_k) \cdot \prod_{\ell_i \in S(w_k)} \left[m[(m-1)!]^{M_i} \cdot \sum_{\{n_t^i(1)\}} \sum_{\in U_k^i} \right.$$

$$\cdot \int_{p_k^i(1)=0}^{1} \frac{[p_k^i(1)]^{1+n_k^i(1)} [1-p_k^i(1)]^{N_k-n_k^i(1)} (N_k)!}{[n_k^i(1)]!\ [N_k-n_k^i(1)]!} \cdot dn_k^i(1)$$

$$\cdot \int_{p_k^i(2)=0}^{1-p_k^i(1)} dp_k^i(2) \ldots\ldots \int_{p_k^i(m)=0}^{1-p_k^i(1)-p_k^i(2)\ldots p_k^i(m-1)} dp_k^i(m)$$

$$\cdot \prod_{\substack{w_t \in W(\ell_i) \\ \neq w_k}} \int_{p_t^i(1)=0}^{1} \frac{[p_t^i(1)]^{n_t^i(1)} [1-p_t^i(1)]^{N_t-n_t^i(1)} (N_t)!\ dp_t^i(1)}{[n_t^i(1)]!\ [N_t-n_t^i(1)]!}$$

$$\left. \int_{p_t^i(2)=0}^{1-p_t^i(1)} dp_t^i(2) \ldots\ldots \int_{p_t^i(m)=0}^{1-p_t^i(1)-\ldots p_t^i(m-1)} dp_t^i(m) \right] \quad \ldots\ldots(6.10)$$

where

M_i = Number of classes below node ℓ_i.

In Equ.(6.10) U_k^i refers to the set of sample counts, $\{n_t^i(1),\ w_t \in W(\ell_i)\}$ such that,

$$\sum_{w_t \in W_o(\ell_i)} n_t^i(1) \quad \geq \quad \sum_{w_t \in W_1(\ell_i)} n_t^i(1) \qquad \text{if } w_k \in W_o(\ell_i)$$

or

$$\sum_{w_i \in W_o(\ell_i)} n_t^i(1) \quad < \quad \sum_{w_t \in W_1(\ell_i)} n_t^i(1) \qquad \text{if } w_k \quad W_1(\ell_i)$$

Each of the multiple integrals in the last product form in Equ.(6.10) reduces to a Beta function, viz.

$$\frac{1}{(m-2)!} \int_{p_t^i(1)=0}^{1} \frac{(N_t)!}{[n_t^i(1)]!\ [N_t - n_t^i(1)]!} [p_t^i(1)]^{n_t^i(1)} \cdot [1-p_t^i(1)]^{N_t - n_t^i(1)+m-2} \cdot dp_t^i(1)$$

$$= \frac{1}{(m-2)!} \quad \frac{(N_t)!}{(N_t+m-1)!} \quad \cdot \frac{(N_t - n_t^i(1)+m-2)!}{(N_t - n_t^i(1))!}$$

Similarily the multiple integral for class w_k is evaluated as,

$$\frac{1}{(m-2)!} \quad \frac{(N_k)!}{(N_k+m-1)!} \quad \frac{(N_k - n_k^i(1)+m-2)!}{(N_k - n_k^i(1))!} \cdot \frac{n_k^i(1)+1}{N_k+m}$$

Substituting these expressions in Equ.(6.10) yields the final expression for the mean accuracy of a binary decision tree for an M class problem, viz.

$$\bar{P}_{cr}(T,M) = \sum_{k=1}^{M} P(w_k) \cdot \prod_{\ell_i \in S(w_k)} m\cdot(m-1)^{M_i} \cdot \left[\sum_{\{n_t^i(1)\}} \cdots \sum_{\in U_k^i} \prod_{w_t \in W(\ell_i)} \frac{(N_t)!\ (N_t - n_t^i(1)+m-2)!}{(N_t+m-1)!\ (N_t - n_t^i(1))!} \cdot \frac{n_k^i(1)+1}{N_k+m} \right] \qquad \ldots\ldots\ldots(6.11)$$

The above expression for the mean accuracy is too complex for direct interpretation. However, for small values of sample sizes, N_t, Equ.(6.11) can be written in a simpler form. In particular, we shall consider the case where there is only a single sample available per class, i.e.

$$N_1 = N_2 = \ \ldots \ \ N_M = 1$$

Since each variable $n_t^i(1)$ can only take on values 0 or 1, the multiple summation over U_k^i in Equ.(6.11) can be replaced by a double summation in the following manner. If there are M_i classes under node ℓ_i, then there are $M_i/2$ classes (and hence samples) from the sets of classes $W_o(\ell_i)$ and $W_1(\ell_i)$, respectively. Assume that class w_k is in the set $W_o(\ell_i)$, i.e.

it is below the left of the two nodes below ℓ_i. Let j_1 and j_2 denote the number of samples from $W_o(\ell_i)$ and $W_1(\ell_i)$ respectively for which feature i was in state 1, i.e. $n_t^i(1)=1$. Then, the rule at ℓ_i would be,

if $j_1 \geq j_2$ go left

if $j_1 < j_2$ go right .

A sample from w_k would be correctly classified only if $j_1 \geq j_2$. Hence the summation within the square brackets in Equ.(6.11) can be written as

$$\sum_{j_1=0}^{M_i/2} \sum_{j_2 \leq j_1} \frac{[(m-1)!]^{M_i-j_1-j_2}[(m-2)!]^{j_1+j_2}}{[m!]^{M_i}} \left[\binom{\frac{M_i}{2}-1}{j_1-1}\binom{\frac{M_i}{2}}{j_2}\frac{2}{m+1} + \binom{\frac{M_i}{2}-1}{j_1}\binom{\frac{M_i}{2}}{j_2}\frac{1}{m+1} \right]$$

In the above equation, the first term is the case when $n_k^i(1)=1$, while the second term is the case, $n_k^i(1)=0$. Upon simplification, it yields

$$\frac{1}{m+1} \cdot \sum_{j_1=0}^{M_i/2} \sum_{j_2 \leq j_1} [(m-1)!]^{M_i-j_1-j_2} \cdot [(m-2)!]^{j_1+j_2} \binom{M_i/2}{j_1}\binom{M_i/2}{j_2} \frac{M_i + 2j_1}{M_i} \quad \ldots.(6.12)$$

Substitution of Equ.(6.12) into Equ.(6.11) will not yield the correct value for the mean accuracy, because the case $j_1 = j_2$ has been treated unsymmetrically in (6.12). Hence, to equalize the correct recognition rates for classes on either side of ℓ_i, we assume the summation over j_2 to be from 0 to $M_i/2$ as for j_1, and then put a weighting factor of 0.5 for the entire

double summation. With this modification, the mean accuracy for the case $N_k = 1$, becomes, after some simplification,

$$\bar{P}_{cr}(T,M) = \sum_{i=1}^{L} \frac{m}{m+1} \cdot \left(\frac{m-1}{m}\right)^{M_i} \sum_{j_1=0}^{M_i/2} \sum_{j_2=0}^{M_i/2} \binom{M_i/2}{j_1} \binom{M_i/2}{j_2} g(j_1,j_2)$$

where

$$g(j_1,j_2) = \frac{1}{2} \frac{1+2j_1/M_i}{(m-1)^{j_1+j_2}} \quad \text{if } j_1 \geq j_2$$

$$= \frac{1}{2} \frac{1+2j_2/M_i}{(m-1)^{j_1+j_2}} \quad \text{if } j_1 < j_2$$

and

$$L = \log_2 M = \text{Levels in the tree}$$

$$M_i = 2^i \qquad (6.13)$$

Experimental Results:

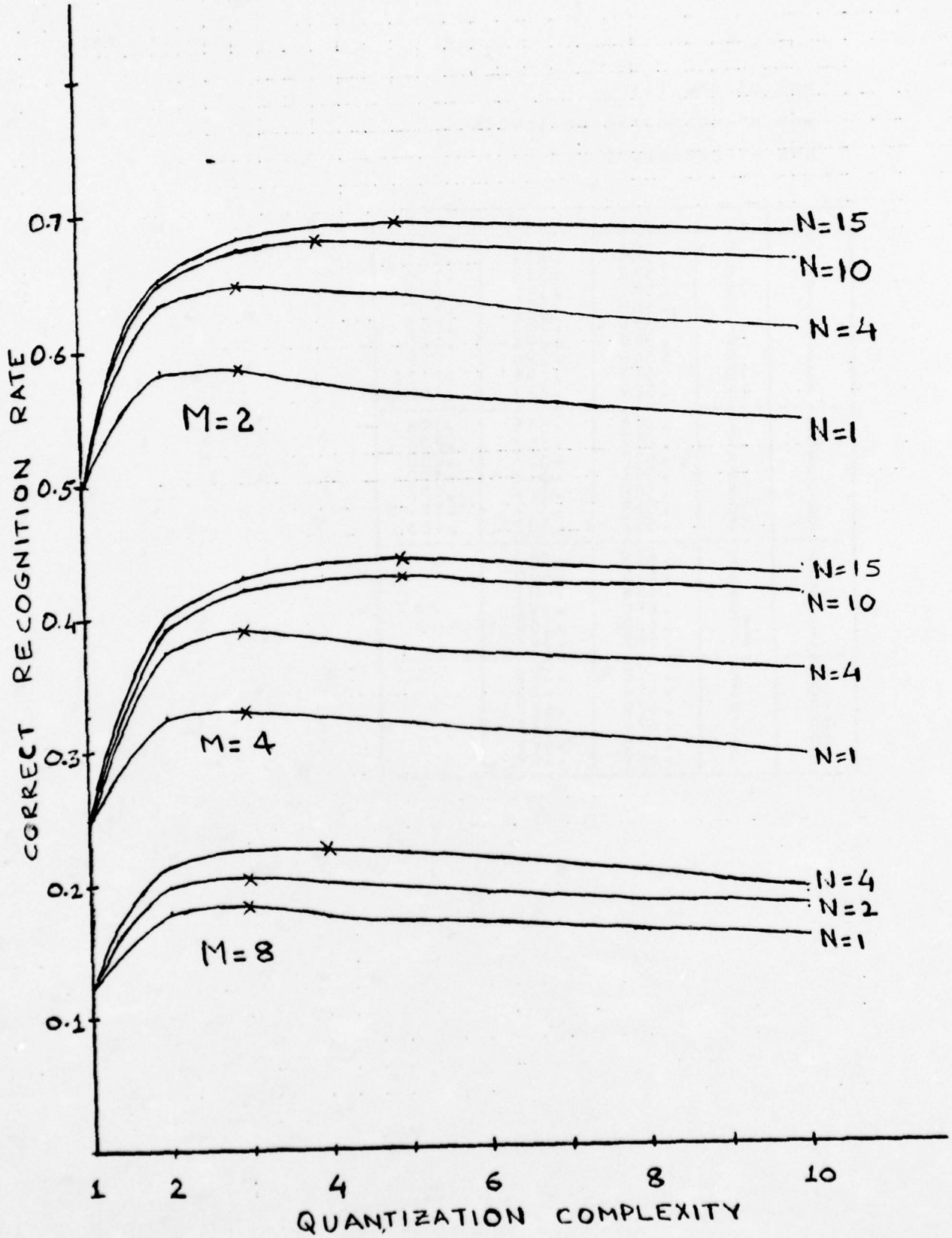
The table below shows the variation of the mean accuracy, Pcr(T,M), with sample size, N, quantization complexity m, and the number of classes M. For any given sample size, it is seen that there is an optimal quantization complexity. This optimal complexity increases with increasing sample size. Moreover, for a given sample size, the optimal complexity for an M-class problem is greater than or equal to the optimal complexity of a M′-class problem, if M > M′. Thus, for the case, N=5, the optimal complexity for a 2-class problem is 3, while for the 4-class case, it is 4. The maximum values of performance in each case are underlined in the table. Graph 2 shows the variation of tree performance with the complexity for various sample sizes N and number of classes M.

N=# of samples

m=# of quantization levels

M=# of classes .

N	m	M=2	M=4	M=8
1	2	.583	.328	.179
1	3	.583	.329	.181
1	4	.575	.323	.176
1	5	.567	.315	.171
1	6	.560	.308	.167
1	7	.554	.303	.163
1	8	.549	.298	.160
1	9	.544	.294	.157
1	10	.541	.291	.155
1	15	.529	.279	.146
2	2	.611	.355	.198
2	3	.617	.362	.203
2	4	.610	.357	.200
2	6	.594	.342	.190
2	8	.580	.329	.182
2	10	.569	.319	.174
2	15	.552	.302	.163
5	2	.639	.382	
5	3	.655	.400	
5	4	.654	.401	
5	5	.650	.397	
5	10	.620	.369	
5	15	.598	.347	
10	2	.652	.395	
10	3	.674	.421	
10	4	.679	.427	
10	5	.678	.428	
10	6	.676	.425	
10	10	.659	.409	


CORRECT RECOGNITION RATE
0.7
0.6
0.5
0.4
0.3
0.2
0.1
N=15
N=10
N=4
N=1
M=2
N=15
N=10
N=4
N=1
M=4
N=4
N=2
N=1
M=8
1
2
4
6
8
10
QUANTIZATION COMPLEXITY

GRAPH 2

6.3. Hierarchical Classifier Versus One-Step Classifier

This section compares the performances of a hierarchical classifier and a one-step classifier for a particular problem, viz., a given set of parameters, $\{p_k^i(j)\}$. Though it is difficult to use the analysis presented here to make any judgements about all M-class problems, one can make the following qualitative statement: if the sample size is small and the error rate of a one-step scheme is only slightly better than a hierarchical scheme on an independent test set, then it is probably better to use the hierarchical scheme. This result is a consequence of the fact that in a decision tree, each node decision is simpler than the M-way decision in a one-step method. Hence, the former requires fewer parameters to be estimated (discrete features assumed here). The result is counterintuitive since in theory(i.e. assuming perfect knowledge of all parameters), no scheme can do better than a one-step classification rule (such as a maximum likelihood rule) which uses ALL features to reach a decision.

Consider the two alternative classification schemes shown in Fig. 6.3a-b below, for a 4-class problem. Two features (discrete) are available and assumed to be class-conditionally statistically independent. Since the one-step scheme in Fig. 6.3b uses both f1 and f2 to make the decision, while the tree uses only a single feature at every node, one would expect the former to perform better. After deriving expressions for the performances of these methods, we illustrate by example, that for small sample sizes, the tree may have a lower error rate.

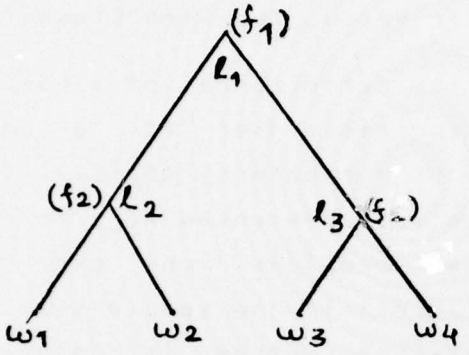


Fig. 6.3a

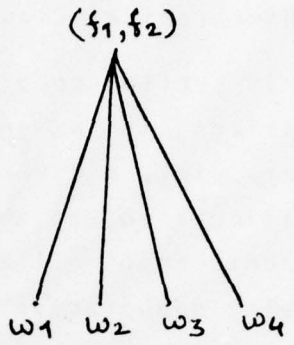


Fig. 6.3b

From the analysis of the preceding section, the performance of a decision tree averaged across the sampling distribution is given by,

$$E'[\ Pc(T,M/\{p_k^i(j)\},\{n_k^i(j)\})\ ]$$

where E' denotes the expectation operator over the distribution of the counts, $\{n_k^i(j),\ j\in(1,m),\ k=1,2,3,4,\ i=1,2\}$. Note that we do not average over the parameters, $\{p_k^i(j)\}$ since we are considering a particular problem. With this modification, and an identical analysis as in the last section, we obtain,

$$\bar{P}c(T,M/\{p_k^i(j)\}) = \sum_{k=1}^{M} P(wk) \cdot \prod_{S(w_k)} \Big\{ \sum_{j=1}^{m} \sum \cdots \sum_{\{n_t^i(j)\}\in I_k^i} p_k^i(j) \cdot \prod_{\omega_t \in W(l_i)} Pr[n_t^i(j)] \Big\} \quad \ldots\ldots(6.14)$$

where S(wk) are the nodes li, on the path to class wk, I_k^i denotes the set of counts $\{n_\ell^i(j)\}$ such that,

$$\sum_{\omega_t \in W_0(l_i)} n_t^i(j) \geq \sum_{\omega_t \in W_1(l_i)} n_t^i(j) \quad \text{if } wk \in W0(li)$$

$$\sum_{\omega_t \in W_0(l_i)} n_t^i(j) < \sum_{\omega_t \in W_1(l_i)} n_t^i(j) \quad \text{if } wk \in W1(li)\ ,$$

and $Pr[n_t^i(j)]$ denote the marginal probabilities of the counts $n_t^i(j)$ which is a binomial distribution given by

$$Pr[n_t^i(j)] = \frac{Nt!}{[n_t^i(j)]!\ [Nt-n_t^i(j)]!} [p_t^i(j)]^{n_t^i(j)} [1-p_t^i(j)]^{N_t - n_t^i(j)}$$

For the one-step rule, the decision is to classify a sample X whose components are x1=j1, and x2=j2, into class wk if,

$$n_K^i(j1).n_K^i(j2) = \underset{1 \le t \le M}{\text{Max}} \{ n_t^i(j1).n_t^i(j2) \} ,$$

and decide ties arbitrarily.

From the above rule, and from a generalization of the case where a single feature is used (such as equ. 6.6), it follows that the performance of the one-step rule averaged over the sampling distribution, is given by,

$$\bar{P}c(OS,M/\{p_k^i(j)\}) = \sum_{k=1}^{M} (wk) \left[\sum_{j=1}^{m} \sum_{j_2=1}^{m} \delta_k(j1,j2).p_k^1(j1).p_k^2(j2). \right.$$
$$\left. \cdot \prod_{t=1}^{4} Pr[n_t^1(j1)].Pr[n_t^2(j2)] \right] \quad \ldots\ldots\ldots\ldots (6.15)$$

where, $\delta_k(j1,j2) = 1$ if,

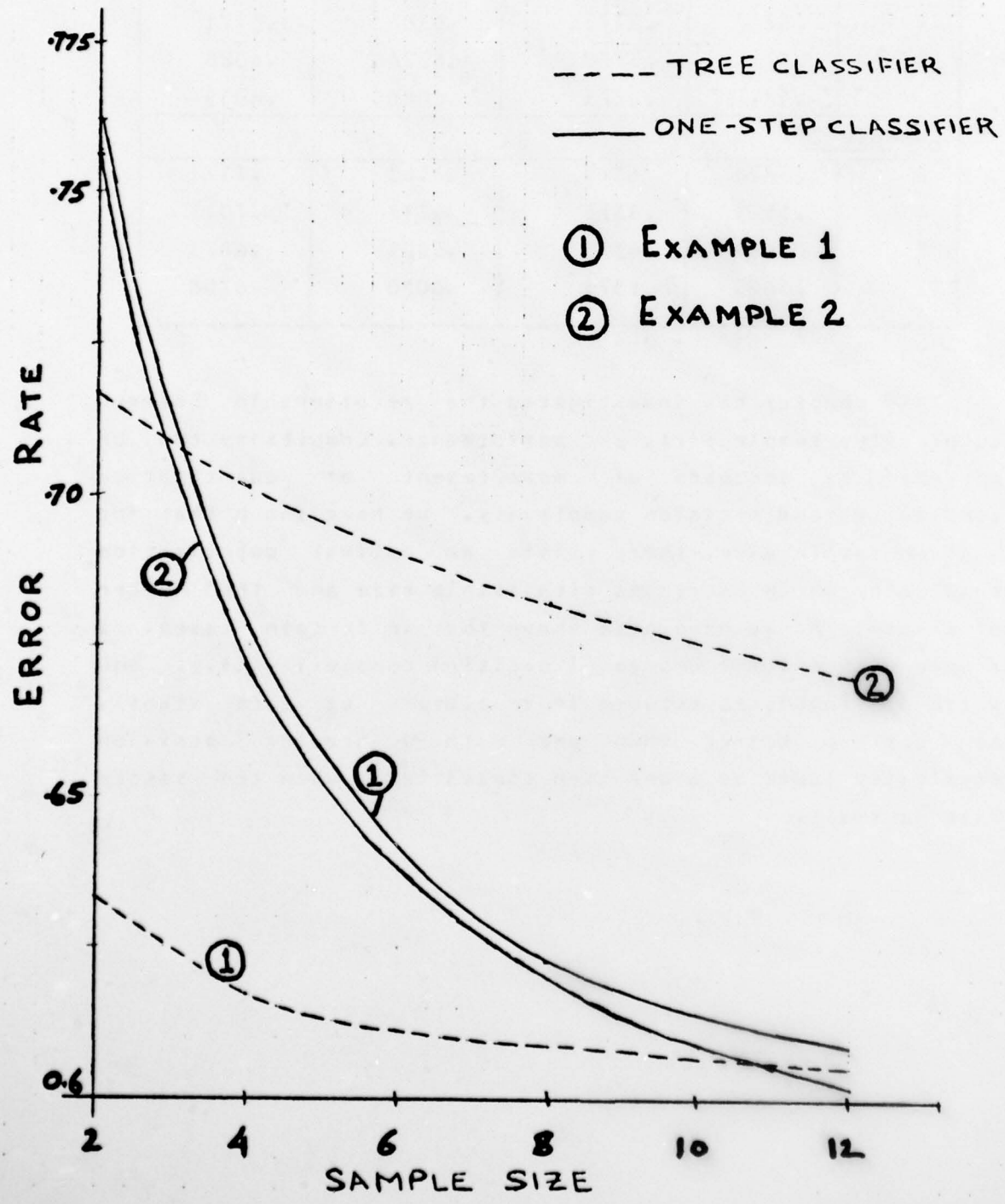
$$n_K^1(j1).n_K^2(j2) = \underset{1 \le t \le 4}{\text{Max}} \{n_t^1(j1).n_t^2(j2)\}$$

and $\delta_k(j1,j2) = 0$ otherwise.

Experimental Results:

Equations (6.14) and (6.15) were used to determine the performances of the tree and one-shot schemes as a function of sample size. The table below shows the error rate for the

various cases considered. In the first example, the tree does better than the one-shot method for all sample sizes, though as sample size becomes large, the latter begins to catch up. Example 2 shows the situation as sample size becomes large. In this case, for sample size less than 4, the tree does better, though as sample size increases, the one shot does better. For infinite sample size, one would expect the one-shot to be consistently better in all cases. Graph 3 depicts the variation of the error rate with sample size for both these examples.


.775
.75
.70
.65
0.6
ERROR RATE
2
4
6
8
10
12
SAMPLE SIZE
TREE CLASSIFIER
ONE-STEP CLASSIFIER
① EXAMPLE 1
② EXAMPLE 2

Example 1:				
Sample size	Ideal OS Error	Ideal Tree Error	Estimated OS error	Estimated Tree error
2	.57	.5775	.7621	.6336
4	.57	.5775	.6784	.6171
8	.57	.5775	.6226	.6088
12	.57	.5775	.6080	.6053
Example 2:				
2	.5625	.6375	.7563	.7171
4	.5625	.6375	.6712	.7015
8	.5625	.6375	.6201	.6823
12	.5625	.6375	.6020	.6706

This chapter has investigated the relationship between complexity, sample size, and performance. Complexity can be regarded as composed of measurement or quantization complexity, and decision complexity. We have shown that for a given sample size, there exists an optimal quantization complexity which increases with sample size and the number of classes, M. We have also shown that in certain cases, a scheme with a lower degree of decision complexity (i.e. one which distinguishes between fewer classes at each stage), can perform better than one with a greater decision complexity (such as a one-step classifier), when the sample size is small.

7. Main Results and Directions for Further Research

This study has led to the following main results regarding the analysis and design of multistage multiclass pattern recognition schemes:

(1) Most parametric and non-parametric multistage pattern classification schemes used in practise, can be described by the theoretical model analyzed in Chapter 2. Certain concepts of admissibility and optimality developed in heuristic programming, have been extended in this work to multistage statistical pattern classification schemes which trade measurement cost for misclassification risk. The conjecture by some earlier researchers that there may not exist Bayes-admissible search strategies which do not measure all features is disproved in our study.

(2) The two types of search strategies derived for this general model require informed heuristics to improve their search efficiency and to test for termination conditions. We have investigated new methods of obtaining lower and upper bounds on the misclassification risk, given a subset of observations on the test sample. Such methods have been used to derive bounds for discrete features which are conditionally statistically independent, or satisfy a first-order tree dependence. The feasibility of computing bounds on the Euclidean distance or similarity measures for binary vectors, establishes the efficacy of our model in implementing nearest neighbour classification schemes.

Similar 'bounding' techniques can be used for more elaborate models of feature dependence, such as second-order and higher order dependence models. Our analyses assumed that the features

were discrete and non-metric. Much work remains to be done in determining whether bcurds on the risk can be derived for the case of continuous valued features and used in formulating admissible strategies.

(3) We have shown that hierarchical classifiers are a special case of the general state-space model. Our analysis has focused on decision trees whose nodce decisions are statistically independent. We have proved that even under the independence assumption, optimizing the performance of each node classifier individually, does not optimize the overall tree performance.

(4) A new systematic approach to optimal tree design has been presented in this work. This method consists of decomposing the design problem into three phases viz., tree skeleton design, feature assignment to its nodes and node decision policy design. Optimal solutions to each design phase are obtained by using the recursive formulation of dynamic programming.

(5) When the features are discrete and non-metric, the design of the optimal decisicn policy at each node, requires excessive computational resources. Methods of clustering decision rules and efficient techniques of discarding suboptimal sets of rules have been developed in Chapter 5. Further investigation of such methods is needed for design problems wherein the node decisions are interdependent.

(6) Feature ranking and a method of bounding the performance of a partially designed tree, have been proposed in this study, as means of reducing the complexity of the feature assignment problem. Improved bounds using parametric models of feature distributions might aid greatly in reducing the execution time of branch-and-bound algorithms such as the one presented in Chapter 5.

(7) The analysis regarding the effects of finite sample size on the performance of hierarchical classifiers has led to two new results:

(a) For a fixed sample size, there exists an optimal quantization complexity for each feature used in a decision tree used for discriminating M classes. The optimal complexity increases with increasing sample size and the number of classes to be distinguished. Our analysis assumed that all features were quantized into the same number of levels. Based on the observed relationship between the optimum complexity and the number of classes, it is our conjecture that a feature used higher up in a tree should be quantized into more levels than one used further away from the root. An analysis similar to that given in chapter 6 might validate this hypothesis.

(b) For small sample sizes, we have shown analytically that certain multiclass problems are better solved using a hierarchical classifier than a one-step classifier, even though, in theory (i.e. given perfect knowledge of the probability functions), the one-step method would perform better in all cases. We have as yet, no way to characterize problems for which this 'small sample phenomenon' occurs. One would like to have a general set of guidelines on the most effective way of using a finite sample to design a hierarchical classifier.

This dissertation provides a broad spectrum of admissible strategies for multistage multiclass recognition problems, and offers a set of optimization procedures for the automated design of optimal hierarchical classifiers.

8. References

[1] Ball G. H. ('66), "A Comparison of Some Cluster Seeking Techniques", Tech. Rep.#RADC-TR-66-514, SRI, Stanford, Ca.

[2] Bellman R.E., Dreyfus S.E., Applied Dynamic Programming, Princeton University Press, Princeton, N.J., 1962.

[3] Chang C. Y., "Dynamic Programming as applied to Feature Subset Selection in a Pattern Recognition System", IEEE Trans. on S.M.C., Vol.SMC-3,pp.166-171, March, 1973.

[4] Freidman J., "A Variable Metric Decision Rule For Nonparametric Classification", SLAC-PUB-1573, SLAC, Stanford California, April, 1975.

[5] Fu K. S., Sequential Methods in Pattern Recognition and Machine Learning, Academic Press, 1968.

[6] Fukunaga K., Narendra P.M., "A Branch and Bound Algorithm for Computing k-Nearest Neighbors", IEEE Trans. Comp., Vol. C-24, No. 7,July,1974.

[7] Hall P. A. V.,"Branch-and Bound and Beyond", Proc. of Second Joint International Conference on Artificial Intelligence, 1971.

[8] Kanal L., "Patterns in Pattern Recognition:1968-1974", IEEE Trans. on Info. Theory, November, 1974.

[9] Knuth D. ('71), "Optimum Binary Search Trees", Acta Informatica, Vol. 1, pp. 14-25 .

[10] Meisel W.S., Michalopoulos D.A., "A Partitioning Algorthm with Application in Pattern Classification and the Optimization of Decision Trees", IEEE Trans. on Computers, Vol. C-22, pp.93-103, January 1973.

[11] Nadler M.,"Error and Reject Rates in a Hierarchical Pattern Recognizer", IEEE Trans. Comp., Vol.C-20,

December, 1971.

[12] Hart P.E., Nilsson N.J., Raphael B., "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", IEEE Trans. of Systems Science and Cybernetics, July, 1968.

[13] Stoeffel J. C., "On Discrete Variables In Pattern Recognition", Ph.D Thesis, Syracuse Univ., 1972.

[14]Wu C., Landgrebe D., Swain P., "The Decision Tree Approach to Classification, TR-EE 75-17, Purdue Univ., May, 1975.

[15] Bell D. A., "Decision Trees In Pattern Recognition", Comp.Sc. TM66,National Physical Laboratory,U.K.,1974.

[16] Mattson R.L., Dammann J.E., "A Technique For Detecting and Coding Subclasses in Pattern Recognition Problems", IBM Journal, July 1965.

[17] Kanal L., Chandrasekaran B., "On Dimensionality and Sample Size In Statistical Pattern Recognition", Proc. NEC, 2-7, 1968.

[18] Hart P., "Searching Probabilistic Decision Trees", AI Group Tech. Note No. 2, SRI Project 7494, Stanford Research Inst. Stanford, California, 1969.

[19] Ball G.H.,"A Comparison Of Some Cluster Seeking Techniques", SRI Tech. Rep. No. RADC-TR-66-514, November, 1966.

[20] Mucciardi A.N., Gose E. E.,"A Comparison Of Seven Techniques for Choosing Subsets of Pattern Recognition Properties", IEEE Trans. Comp. Vol. C-20, September 1971.

[21] Fukunaga K., Introduction To Statistical Pattern Recognition, Academic Press, 1972.

[22] Winston P.,"A Heuristic Program That Constructs Decision Trees", MIT Project MAC, AI Memo# 173, 1969.

[23] Chow C., Liu C.," Approximating Discrete Probability

Distributions and Dependence Trees", IEEE Trans. on Inf. Theory, Vol. IT-14,No. 3, pp.462-467, 1968.

[24] Reinwald L.T., Soland R.M., "Conversion of Limited Entry Decision Tables To Optimal Computer Programs I: Minimum Average Processing Time", JACM Vol. 13, pp.339, 1966.

[25] Reinwald L.T., Soland R.M., "Conversion of Limited Entry Decision Tables To Optimal Computer Programs II: Minimum Storage Requirements", JACM Vol. 14, pp.742, 1967.

[26] Pollack S.L.,Hicks H.T., Harrison W.J., Decision Tables- Theory and Practice, Wiley Interscience, New York, 1971.

[27] Gaffey W.R.," Discriminatory Analysis:Perfect Discrimination As The Number of Variables Increases", Rep.No.5 Project No. 21-49-004, USAF School Of Aviation Medicine, Randolf Field, Texas, February 1951.

[28] Wald A., Sequential Analysis, Wiley, New York, 1947.

[29] Watanabe S., Pakvasa ,"Subspace Methods In Pattern Recognition", Proc. of First Joint International Conf. on Pattern Recognition, Washington D.C. 1973.

[30] Therrien C.W., "A Generalized Approach To Linear Methods Of Feature Extraction", Tech. Note 1974-59, Lincoln Lab., MIT, December 1974.

[31] Friedman J., Baskett F., Shustek L.J., "A Relatively Efficient Algorithm For Finding Nearest Neighbours", SLAC-PUB-1448, CS-445, June 1974, Stanford, Calif. 1974.

[32] Whittle P. , Optimization under Constraints- Theory and Applications of Nonlinear Programming, New York Wiley, 1971.

[33] Hughes G.F., "On the Mean Accuracy Of Statistical

Pattern Recognizers",IEEE Trans. IT, Vol.IT-14, pp. 55-63, 1969.

[34] Chandrasekaran B., Jain A., "Optimum Complexity and Independent Measurements", IEEE Trans. Computers, Vol. C-23, No. 1, January 1974.

[35] Cover T.M., "The best two independent measurements are not two best",(Corresp) IEEE Trans. SMC,January 1974.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 77 - 0825	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) OPTIMAL AND HEURISTIC SYNTHESIS OF HIERARCHICAL CLASSIFIERS		5. TYPE OF REPORT & PERIOD COVERED Technical report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Ashok V. Kulkarni		8. CONTRACT OR GRANT NUMBER(s) ENG 73-04099 & AFOSR 76-2901
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Computer Science University of Maryland College Park, Maryland 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS National Science Foundation, Engineering Division 1800 G Street, N.W. Washington, D.C. 20550		12. REPORT DATE August 1976 13. NUMBER OF PAGES 174
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Distribution of this document is unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES Controlling Office Name & Address (as in #11) also: Air Force Office of Scientific Research Bolling AFB, Washington, D.C. 20332		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Multistage schemes such as hierarchical classifiers have been found useful for many multiclass pattern recognition tasks. This dissertation investigates the theoretical properties of a general model of multistage multiclass recognition schemes. The generality of the model allows one to describe a large class of parametric and non-parametric schemes commonly used in terms of the model parameters. Two classes of admissible and optimal strategies for obtaining the optimal decision are analyzed. These strategies employ lower and upper bounds on a risk function to improve the search efficiency. New methods of computing the bounds are

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 77-0825	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) OPTIMAL AND HEURISTIC SYNTHESIS OF HIERARCHICAL CLASSIFIERS		5. TYPE OF REPORT & PERIOD COVERED Interim 6. PERFORMING ORG. REPORT NUMBER TR-469
7. AUTHOR(s) Ashok V. Kulkarni		8. CONTRACT OR GRANT NUMBER(s) AFOSR 76-2901
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Computer Science University of Maryland College Park, Maryland 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research/NM Bolling AFB DC 20332		12. REPORT DATE August 1976 13. NUMBER OF PAGES 174
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Multistage schemes such as hierarchical classifiers have been found useful for many multiclass pattern recognition tasks. This dissertation investigates the theoretical properties of a general model of multistage multiclass recognition schemes. The generality of the model allows one to describe a large class of parametric and non-parametric schemes commonly used in terms of the model parameters. Two classes of admissible and optimal strategies for obtaining the optimal decision are analyzed. These strategies employ lower and upper bounds on a risk function to improve the search efficiency. New methods of computing the bounds are

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE UNCLASSIFIED

20. Abstract (Concluded)

investigated for the cases when the features are class-conditionally statistically independent and where they satisfy a first-order tree dependence relation. Bounds are also derived for use in nearest-neighbor classification schemes employing a Euclidean distance measure and various similarity measures for non-metric feature vectors.

Hierarchical classifiers are special types of multistage recognition schemes wherein at each stage certain classes are rejected from consideration as labels of the test sample. Theoretical properties of decision trees whose node decisions are statistically independent are investigated. Even under this independence assumption the optimal tree design task is a complex one.

A three phase decomposition of the tree design problem is proposed viz. tree skeleton design, feature selection at its nodes and decision function design at each node. Optimal solutions to each design phase are obtained using a dynamic programming formulation.

These optimal design methods rapidly become cumbersome in computational resources as the number of features and classes increase. This study proposes various techniques for reducing the computational complexity incurred in finding the optimal features to be measured at each node and the optimal decision policy. A method of clustering decision rules and rejecting sets of suboptimal rules without evaluating each individual one is proposed. Feature ranking and a branch-and-bound method are described for reducing the possible feature assignments to be considered in finding the optimal feature measurement policy.

In practice, the decision rules at the nodes have to be estimated from a finite set of design samples. This work investigates the relationship between the expected tree performance, sample size and the number of states (quantization levels) of each feature. It is shown that for an M-class recognition scheme using a decision tree, there exists an optimal quantization complexity. The optimum complexity increases with sample size and with the number of classes to be distinguished. For small sample sizes, it is shown that a multistage decision scheme can have a lower error rate than a single stage scheme which uses all the available measurements in an M-way decision rule.